2012

Large-scale Tree Parsimony

Andre Wehe
Iowa State University

Large-scale tree parsimony

by

André Wehe

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-majors: Computer Engineering;

Computer Science

Program of Study Committee:

Oliver Eulenstein, Co-major Professor

Srinivas Aluru, Co-major Professor

David Fernández-Baca

Manimaran Govindarasu

Suraj C. Kothari

J. Gordon Burleigh

Iowa State University

Ames, Iowa

2012

DEDICATION

I would like to dedicate this thesis to my wife Amy. Without her support I would not have been able to complete this work. I would also like to thank my friends and family for their loving guidance and assistance during many years of study and research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, I would like to thank my wife Dr. Amy Wehe. Her words of encouragement have often renewed my strength. I would like to thank both my major professors Dr. Oliver Eulenstein and Dr. Srinivas Aluru for their guidance, patience, and support throughout this research. A very special thank to Dr. J. Gordon Burleigh for his insights in evolutionary biology and financial support in difficult times. I would also like to thank my committee members for their efforts and contributions to this work: Dr. David Fernández-Baca, Dr. Manimaran Govindarasu, and Dr. Suraj C. Kothari. My friends have been there for me to cheer me on and to commiserate with me in tough times. I like to thank my friends in the orchestra for their friendship. And last but not least, I am grateful to my family for instilling in me a desire to learn, believing in me, and encouraging me to pursue my dreams.

ABSTRACT

Finding the tree of life is one of the major challenges that scientists are attempting to solve. It is widely believed that the evolution of species can (mostly) be depicted in a tree graph, the phylogenetic tree. However, the true phylogenetic species tree is often unknown. One approach is to computationally infer phylogenetic trees from phylogenetic information encoded in genomic data. With the advancement of sequencing techniques, we have a rapidly growing availability of phylogenetic data, which enable the construction of large-scale phylogenetic trees. This thesis addresses algorithmic issues for the construction of large-scale phylogenetic species trees, the supertrees, and the exploration and analysis of large-scale phylogenetic trees. We present (i) new algorithms for local search methods for supertree construction that reduce the time complexity by an order of magnitude and a parallelization for these methods, (ii) new methods for constructing better supertrees from estimated trees and inferring small, exact phylogenetic trees, and (iii) a novel, interactive visual method for the large-scale tree exploration and the concurrent analysis of multiple gene trees and one species tree.

CHAPTER 1. General Introduction

All species and organisms are genetically related. This is one of the most profound scientific insights. A vast evolutionary tree, the tree of life, can represent the evolutionary relationships among species and organisms. The estimated 1.7 million living species on earth and their abstruse genetic relationships present scientists with a challenge to construct the tree of life. Trees that describe evolutionary relationships between species are called phylogenetic trees or phylogenies. Modern new sequencing technology provides a rapidly increasing amount of available genomic sequence data, which include phylogenetic information now ready to be harvested for phylogenetic analyses. Many methods and models have been developed for building phylogenies from this kind of information [34]. For various reasons, phylogenies do not have to agree with each other. For instance, the evolution of genes can be different than the evolution of species. Therefore, the next challenge is to find a tree, a so-called supertree, that reflects the evolution of species. The simplified problem is, given a taxon set (that is a set of species classifications) and a collection of discordant phylogenetic input trees on partial taxon sets, infer the correct phylogeny for the taxa set.

This thesis addresses some of the algorithmic, methodological, and visualization challenges posed by this supertree problem. Chapter 2 and 3 address new algorithmis for the local search heuristics that infer phylogenetic trees based on gene duplications. Chapter 4 introduces new methods for creating supertrees and improving estimated supertrees based on Robinson-Foulds [60] distance. Chapter 5 discusses a new interactive visualization method for supertree exploration and analysis of gene duplications, gene losses, and deep coalescence.

1.1 Gene Tree Parsimony

Scientists believe that gene duplications play a major role in evolution [70]. Gene duplication is a phenomenon in which a region of DNA that contains genes duplicated, resulting in two copies of a gene. Before long, both copies may evolve independently and may serve different functions. Methods and tools have been developed for constructing phylogenetic trees for genes (e.g., Maximum Parsimony [66, 69] and Maximum Likelihood [66, 69]), but it is a fallacious assumption that the evolution of genes has to be identical to the evolution of species. Due to complex evolutionary processes, such as gene duplication and gene loss, these phylogenies may differ.

The gene duplication model [37] provides a computational framework for explaining phylogenetic tree differences with gene duplication events, and then the model states that the phylogenetic species tree is the species tree that induces the fewest gene duplication events for a set of phylogenetic gene trees. However, this gene duplication problem is an optimization problem that is NP-hard [46]. Although species trees with very few taxa can be computed, inferring species trees with many taxa becomes computationally intractable. Therefore, one of the best approaches for inferring larger species trees is estimating the trees, for instance with local search heuristics [56, 55, 6, 7]. Although these local search heuristics have proven to work well in practice, their capabilities have been very limited on complex and large-scale analysis. The algorithmic time complexity for the heuristic search prohibits inferring large-scale species trees with many thousands of taxa. Also, data sets with complex phylogenetic gene histories often present a challenge to search heuristics, which then often results in poorly estimated species trees. However, this can often be compensated for by repeatedly estimating species trees until a good tree is found.

Chapter 2 introduces a serial algorithm and Chapter 3 introduces a parallelization that greatly improve the run-time of local search heuristics. Chapter 4 introduces a new method that can improve estimated species trees. Chapter 5 introduces a novel interactive visualization method for exploring and analyzing genomic events in large-scale species and gene trees.

1.2 Robinson-Foulds Supertree

There are various measures to determine the similarity of trees. The Robinson-Foulds (RF) distance [60] is one of the most widely used. In particular, supertrees have been evaluated by measuring the RF distance between supertrees and a collection of input trees (e.g., [77, 15, 32]). In fact, recently, new supertree methods have been developed that seek a supertree that altogether minimizes the RF distance [5, 23] between the supertree and a collection of input trees. Like many other supertree problems, including the gene duplication problem, the RF supertree problem is NP-hard [33]. Similar to the gene duplication problem, the RF supertree problem has also been successfully addressed by local search heuristics [5, 23].

Simplified, the RF distance for a pair of rooted trees measures the number of clusters (or clades) that are not shared in the pair of trees. Thus, in the rooted setting, the RF supertree problem seeks a rooted binary supertree with the minimum sum of RF distances of all tree pairs that consist of the species tree and an input tree.

Chapter 4 introduces new RF supertree problems that seek minimum RF supertrees in a restricted search space. These new problems can be solved exactly in reasonable time even for larger supertrees. Now, by harvesting the knowledge of estimated RF supertrees, this method can greatly improve the estimated RF supertrees.

1.3 Tree Reconciliation

One phenomenon in which scientists are interested is the evolution of genes within species. Often the evolutionary history of genes is incongruent with the evolution of species. The gene/species tree reconciliation explains this incongruence with genomic events. Specifically, the reconciliation is a map between a gene tree and a species tree with gene duplications and losses being placed to explain any incongruence between the trees.

The most widely used techniques for displaying and analyzing small reconciliations are reconciled trees and embedded reconciled trees. However, these standard visualization techniques can grow quadratic in size with the number of taxa. Thus, complex, larger-scale reconciliations become prohibitively large, and detailed analysis becomes unwieldy in practice.

Chapter 5 introduces a novel visualization method that uses interactive techniques for the exploration and analyzis of complex, large-scale reconciliations.

1.4 Thesis Organization

This thesis includes of six chapters. Chapter 2 introduces a new algorithm and a new lightweight heuristic search strategy. Both greatly speed up the local search for the NP-hard gene duplication problem, enabling for the first time supertree construction of up to 100,000 taxa. We presented this research at BICOB 2010 [74], and the chapter contains a full and revised report of this research. For the same gene duplication problem, we describe an efficient parallelization in Chapter 3. This paper was published in JPDC [76]. Chapter 4 introduces new methods and RF supertree problems. This paper has been accepted for publication at ISBRA 2012 [75]. Chapter 5 discusses a novel interactive visualization method designed for exploring and analyzing complex, large-scale gene/species tree reconciliations. This is new research and has not been published yet. Concluding remarks appear in Chapter 6.

1.4.1 Authors' Contributions

Chapter 2: André Wehe was responsible for algorithm design, writing major parts of the manuscript, and for the program implementation; Dr. J. G. Burleigh was responsible for the experimental data sets, and he contributed to the writing of the manuscript.

Chapter 3: André Wehe was responsible for algorithm design, writing major parts of the manuscript draft, and for the program implementation, and he contributed to the experiments; Wen-Chieh Chang was responsible for the experiments and the experimental section of the manuscript, and he contributed to algorithm design, program implementation, and the writing of the manuscript; Dr. Srinivas Aluru contributed to revising and writing major parts of the manuscript.

Chapter 4: André Wehe was responsible for algorithm design, writing major parts of the manuscript, and for the program implementation, and he contributed to major parts of the experiments; Dr. J. G. Burleigh was responsible for the experiments, and he contributed to

writing of the manuscript; Dr. O. Eulenstein contributed by revising and writing major parts of the manuscript.

Chapter 5: André Wehe was responsible for the visual method design, the software design, implementation, and writing of major parts of the manuscript; Dr. J. G. Burleigh contributed to the visual method design and testing of the software; Dr. O. Eulenstein contributed to revising and writing of the manuscript.

CHAPTER 2. Scaling the Gene Duplication Problem towards the Tree of Life: Accelerating the SPR Heuristic Search

Modified from a paper published in the International Conference on Bioinformatics and Computational Biology, 2010

André Wehe and J. Gordon Burleigh

Abstract

The gene duplication problem seeks the species tree that results in the fewest duplications across a collection of gene trees. We describe two new improvements to existing heuristics that greatly accelerate the SPR local search. The improvements involve (i) reducing the computational complexity by $\Theta(n/h)$ for sequential evaluations of the reconciliation cost on n taxa and a tree height of h, and (ii) a lightweight and effective heuristic search strategy. In analyses of both empirical and simulated data sets, the new improvements display tremendous speedup from the best existing heuristics with no apparent loss of accuracy. We also demonstrate with simulated data that the improved heuristics can easily compute species trees with 100,000 taxa on a single desktop computer.

2.1 Introduction

Phylogenetic trees can be powerful tools for examining patterns of biodiversity [25] and the structure of communities and ecosystems [30] as well as comparative analyses of species and character evolution and diversification [43, 59]. Performing such analyses on a macroevolutionary or ecological scale often requires extremely large phylogenies. There is great

interest in developing methods that can synthesize new molecular data into extremely large-scale phylogenetic hypotheses, and ultimately the tree of life. However, large-scale phylogenetic inference presents several enormously challenging computational problems, including the limitations of available memory and the computational burden of efficient heuristics [3]. In this paper, we present two new approaches that greatly speed up the inference of large-scale phylogenetic trees based on gene tree/species tree reconciliation, allowing phylogenetic analyses on an unprecedented scale.

Maximum parsimony (MP) and maximum likelihood (ML) are among the most popular methods for inferring trees. Both take an alignment of gene sequences and attempt to identify the optimal tree based on an optimality criterion, either the tree that implies the fewest character substitutions (MP) or the tree that maximizes the probability of observing the data based on a model of evolution (ML). Both MP and ML methods have been used to build phylogenetic trees with tens of thousands of taxa (e.g., [51, 36]). However, these methods have some limitations for large-scale phylogenetics. First, the size of the character matrices can create an enormous memory burden that, in practice, limits the number of genes that can be incorporated into an analysis. For example, if the average gene is conservatively 1000 base pairs long, a character matrix with 100,000 taxa and just 10 genes would include 1 billion cells and 1000 genes would be 100 billion cells. Second, both MP and ML implicitly assume that all of the genes share a common evolutionary history. However, evolutionary processes such as gene duplications and losses, incomplete lineage sorting (or deep coalescence), horizontal transfer, and recombination can affect the topologies of gene trees, producing incongruence between the gene tree and species tree topologies [47]. This incongruence among gene trees can produce error in MP and ML analyses (e.g., [11, 58]).

One approach to inferring species trees from genes with complex evolutionary histories is gene tree parsimony (GTP), which seeks to identify the species tree that implies the minimum number of events that cause conflict among the gene trees (e.g., [37, 47]). Specifically, GTP takes a collection of gene trees and seeks a species tree that contains all taxa represented in the gene trees and implies the fewest gene duplications, duplications and losses, or deep coalescence events (e.g., [39]). Thus, unlike MP or ML, GTP does not assume a single underlying phylogeny

for gene trees. Furthermore, since the input for GTP analysis is a set of trees rather than gene alignments from each gene, GTP has a much lower memory burden than MP or ML. However, GTP is an NP-hard problem (e.g., [46]), and analyses with more than ≈10 taxa require heuristic solutions.

Recent improvements in the speed of local search heuristics have allowed GTP to be applied to genome-scale data sets with hundreds of taxa (e.g., [6, 4]). For example, a recent GTP analysis included 18,896 gene trees from 131 plant taxa [18]. Yet this is still far from the tree of life.

Our contribution: We present two new improvements to GTP heuristics that in practice produce enormous speedup in the existing heuristic with no apparent loss of accuracy. These new heuristic approaches enable GTP analyses to incorporate large sections of the tree of life from large proportions of the genome.

Specifically, the algorithm we developed is useful for the local search heuristic based on the SPR operation. The local search heuristic is a step-wise process of optimizing a species tree, where the SPR defines a local search space which is evaluated for a species tree with a better reconciliation cost (in our case, a lower gene duplication cost). The tree rearrangement operation SPR (subtree prune and regraft [1, 17]) is informally defined as follows: it relocates a clade into a different edge or to the root in a rooted tree. The local SPR neighborhood is the set of trees that can be reached by one SPR operation, thus all trees in a local search neighborhood can be constructed by sequentially applying SPR operations.

For the gene duplication model by [37] the reconciliation costs is inferred from a node-mapping (we will later refer to it by LCA-mapping) between gene trees and species trees. A single SPR operation on the species tree almost always changes this mapping, thus it is required to recompute the mapping frequently throughout an heuristic search. In this paper, we show that in practice it is likely that most node-mappings remain unchanged and only a minor subset of node-mappings is actually altered. This is due to the fact that multiple gene duplications occur throughout the species tree, and neither the species tree nor guidance tree is likely to completely pectinate.

In Section 2.3 we present a new algorithm that allows to efficiently identify only those

mappings that actually get altered, thus allowing to quickly update the mapping from a previous mapping of species tree with a one-SPR distance. This results in a significant speed-up for the SPR local search heuristics, which entails the frequent processing of sequential species trees of one-SPR distance.

In Section 2.5 we show a lightweight SPR heuristic search strategy that also allows for a significant speed-up. This lightweight heuristic search diverges from the the classical SPR heuristic search strategy by not following the steepest descending path. Instead, the search follows the first descending path strategy, which proceeds with the first better optimal species trees discovered and not the locally optimal species tree. In Section 2.6 we show that both improvements lead to a significant speed-up and an equal or better species tree. This allows the reconstruction of species trees of up to 100,000 species.

2.2 Basic Notations and Preliminaries

The following conventions are used for convenience throughout this chapter. The notation was inspired by the notation used by O. Eulenstein and M. S. Bansal (e.g., [7]). Let T be a rooted tree. We denote the vertex set, edge set, and leaf set of T by $E(T)$, $V(T)$, and $\mathsf{Le}(T)$ respectively. The root of T is denoted by $\mathsf{Rt}(T)$. Given a vertex $v \in V(T)$, we denote the children of v by $\mathsf{Ch}_T(v)$, and the parent of v by $\mathsf{Pa}_T(v)$. Two vertices in T are called *siblings* (of each other) if they have the same parent. We write (u, v) to denote the edge $\{u, v\} \in E(T)$ where $u = \mathsf{Pa}_T(v)$. The set of *internal vertices* of T, denoted $I(T)$, is defined to be $V(T) \setminus \mathsf{Le}(T)$. We call T *full binary* if each vertex $v \in I(T)$ has exactly two children. Throughout this chapter the term tree refers to a rooted, fully binary tree.

Given a vertex-set U, we denote by $T(U)$ the unique subtree of T that spans $U \cap V(U)$ with the minimum number of vertices. Furthermore, the *restriction* of T to U, denoted by $T_{|U}$, is the tree that is obtained from $T(U)$ by suppressing all non-root vertices of degree two. The *subtree* of T rooted at $v \in V(T)$, denoted by T_v, is defined to be $T_{|U}$, for $U := \{u \in \mathsf{Le}(T) \mid v \text{ is on the shortest path between } \mathsf{Rt}(T) \text{ and } u\}$. We define $x \leq_T y$ to be the partial order on $V(T)$, where $x \in V(T)$ and $y \in V(T_x)$.

Let A and B be two sets; we write the symmetric difference of both sets as $A \Delta B$.

The gene duplication model by [37] allows to determine gene duplications and their locations in the species trees. In Section 2.4 we define in more detail the method used by this model for inferring gene duplications. Computationally, this model relies on a mapping between the species trees and the gene trees based on the LCA (Least Common Ancestor) of species. Thus, for our algorithm we focus first exclusively on this LCA mapping.

Definition 2.2.1. [LCA]

The *least common ancestor* of a non-empty subset of nodes $L \subseteq V(T)$ in a tree T, denoted by $lca_T(L)$, is the unique smallest lower bound of L under $\leq_T$.

Definition 2.2.2. [Mapping]

The leaf-mapping $\mathcal{L}_{G,S} \colon \mathsf{Le}(G) \to \mathsf{Le}(S)$ maps a leaf node $g \in \mathsf{Le}(G)$ to a leaf node $s \in \mathsf{Le}(S)$. The extension $\mathcal{M}_{G,S} \colon V(G) \to V(S)$ of $\mathcal{L}_{G,S}$ is the mapping defined by

$$\mathcal{M}_{G,S}(g) := \begin{cases} \mathcal{L}_{G,S}(g) & g \in \mathsf{Le}(G) \\ lca_S(X), X = \{\mathcal{M}_{G,S}(v) \,|\, v \in V(G_g)\} & \text{otherwise.} \end{cases}$$

Definition 2.2.3. [Comparability]

Given trees G and S, we say that G is comparable to S if, for each $g \in \mathsf{Le}(G)$, the leaf-mapping $\mathcal{L}_{G,S}(g)$ is well defined.

Definition 2.2.4. [Planted tree]

Given a tree T, the planted tree $\Psi(T)$ is the tree obtained by adding a root edge $(r, \mathsf{Rt}(T))$ to T, where $r \notin V(T)$.

Next, we formally define the tree rearrangement operation SPR [1, 17] and the SPR^*, which is a special type of SPR where the subtree is rearranged at the root of the tree.

Definition 2.2.5. [Rooted SPR operation]

Let T be a tree, $e \in E(T)$, where $e = (u, v)$ and $u = \mathsf{Pa}_T(v)$, and X, Y be the connected components that are obtained by removing edge e from T such that $v \in X$ and $u \in Y$. We define $SPR_T(v, y)$ (Figure 2.2.1 shows examples) for $y \in Y$ to be the tree that is obtained from $\Psi(T)$ by first removing edge e, and then adjoining a new edge f between v and Y as follows:

1. Create a new node y' that subdivides the edge $(\mathsf{Pa}_T(y), y)$.

2. Add edge f between nodes v and y'.

3. Suppress the node u, and rename y' as u.

4. Contract the root edge.

We say that the tree $SPR_T(v, y)$is obtained from T by a subtree prune and regraft (SPR) operation that prunes subtree T_v and regrafts it above node y (see Figure 2.2.1).

The operation $SPR^*_T(v, y)$ is an $SPR_T(v, y)$ operation, such that either $y = \mathsf{Rt}(T)$ or $\mathsf{Pa}_T(v) = \mathsf{Rt}(T)$.

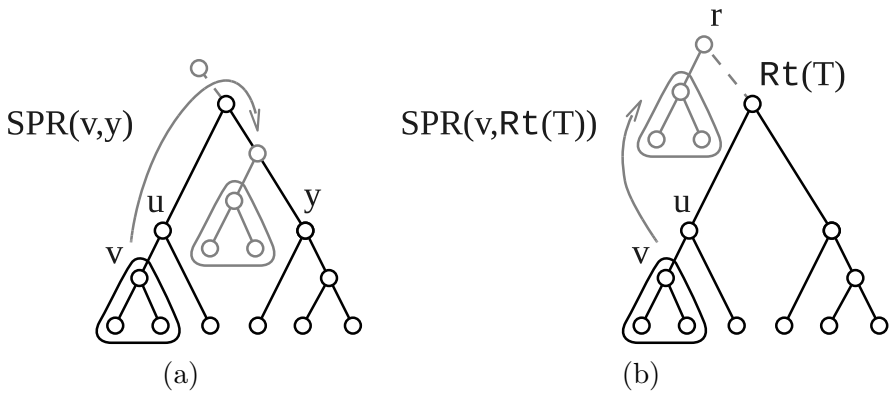


Figure 2.2.1 (a) $SPR_T(v, y)$ on tree T, where the subtree T_v is pruned and regrafted into the edge $(y, \mathsf{Pa}_T(y))$. (b) $SPR^*_T(v, \mathsf{Rt}(T))$ on tree T, where the subtree T_v is pruned and regrafted into the root edge $(r, \mathsf{Rt}(T))$ of $\Psi(T)$.

Remark 2.2.6. Any SPR_T operation can be emulated by 2 successive SPR^*_T operations.

When processing the heuristic search, the species tree changes frequently. Although the gene trees remain unchanged, the species tree changes with every SPR operation, thus the LCA mapping is likely to change for every new species tree. The fastest method for computing the LCA mapping addresses 2 problems. First, we need an efficient way to compute the LCAs for any subset of species in the species tree S. This can be solved in linear time to the species tree size $V(S)$ using the LCA algorithm in [12]. Second, we need to compute the mapping between gene trees and guidance tree. This step is more computationally costly, since the gene tree input size $m := \sum_{i=1}^{k} |V(G_i)|$ for k gene trees usually far exceeds the size of the species

tree $|V(S)|$. The computational complexity for the mapping is linear in m. In practice it can be computed initially quickly even for huge trees, but for the typical frequent repetitions in a heuristic search, the computation of the mapping becomes unwieldy for larger data sets. In theory, all mappings can be altered by a single SPR operation on the species tree. However, in practice it is likely that only a minor subset of mappings, denoted with set L, is actually altered; i.e. $|L| \ll m$. It is therefore justified to consider the following problem.

Problem 2.2.7. [SPR mapping problem]

Instance: Let tree G be comparable to tree S, and let $S' := SPR_S(v, y)$ for some $v, y \in V(S)$.

Problem: Find $\mathcal{M}_{G,S} \Delta \mathcal{M}_{G,S'}$.

2.3 Solving the SPR Mapping Problem

Let S be a tree and G be a comparable tree. Any SPR_S operation can be emulated by 2 successive SPR_S^* operations, thus for simplification we let $S' := SPR_S^*(p, \cdot)$ for some prune subtree S_p. For convenience, we will keep the notation G, S, S', and p throughout this section.

The straightforward solution for solving Problem 2.2.7 is to compute $\mathcal{M}_{G,S}$ and $\mathcal{M}_{G,S'}$, then the difference is $\mathcal{M}_{G,S} \Delta \mathcal{M}_{G,S'} := \left\{ lca_S\left(\mathcal{L}_{G,S}\left(\mathsf{Le}\left(G_g\right)\right)\right) \neq lca_{S'}\left(\mathcal{L}_{G,S'}\left(\mathsf{Le}\left(G_g\right)\right)\right) : g \in V(G) \right\}$. Let $n := |V(S)|$ and $m := |V(G)|$, then the mappings can be computed in linear time [$Bender : 2000 : TLP$], so the run-time for the straightforward solution is $O(m+n)$. However, a problem in computational phylogeny is the repeated computation of $\mathcal{M}_{G,S} \Delta \mathcal{M}_{G,S'}$ where $S' = SPR_S(p, \cdot)$, e.g. [6, 73, 5, 23]. This problem is similar to Problem 2.2.7 with the difference that $\mathcal{M}_{G,S}$ has been already computed.

Problem 2.3.1. [SPR mapping update problem]

Instance: Let tree G be comparable to tree S, let $S' := SPR_S(v, y)$ for some $v, y \in V(S)$, and let $\mathcal{M}_{G,S}$ be given.

Problem: Find $\mathcal{M}_{G,S} \Delta \mathcal{M}_{G,S'}$.

2.3.1 Structural Mapping Properties

Definition 2.3.2. [Blue-node]

Let S be a tree, G be a comparable tree, and $S' := SPR^*_S(p, y)$ for some $p, y \in V(S)$. A tree node u is colored blue (hence, a blue-node) when $\mathcal{M}_{G,S}(u) \neq \mathcal{M}_{G,S'}(u)$.

Definition 2.3.3. [Red-node]

Let S be a tree, G be a comparable tree, and $S' := SPR^*_S(p, y)$ for some $p, y \in V(S)$. A tree node $v \in S$ is colored red (hence, a red-node) when there exists a node $u \in V(G)$ such that $v = \mathcal{M}_{G,S}(u)$ and $\mathcal{M}_{G,S}(u) \neq \mathcal{M}_{G,S'}(u)$.

Remark 2.3.4. A node u as in Definition 2.3.3 is a blue-node.

2.3.1.1 Structural properties of blue nodes in gene trees

Lemma 2.3.5. *[Divided mappings of blue-node's descendants]*

Let S be a tree, G be a comparable tree, and Let $S' := SPR^*_S(p, y)$ for some $p, y \in V(S)$, and let $b \in V(G)$ be a blue-node. Then, there exists $\mathcal{M}_{G,S}(v) \in V(S) \setminus V(S_p)$ such that $b \leq_G v$, and there exists $\mathcal{M}_{G,S}(u) \in V(S_p)$ such that $b \leq_G u$.

Proof. Assume $\nexists v$ such that $\mathcal{M}_{G,S}(v) \in V(S) \setminus V(S_p)$ such that $b \leq_G v$, then $\mathcal{M}_{G,S}(V(G_b)) \subseteq V(S_p)$. Since $S_p = S'_p$, it follows that $lca_S(\mathcal{M}_{G,S}(V(G_b))) = lca_{S'}(\mathcal{M}_{G,S}(V(G_b)))$, thus $\mathcal{M}_{G,S}(b) = \mathcal{M}_{G,S'}(b)$, but then b is not a blue-node.

Assume $\nexists u$ such that $\mathcal{M}_{G,S}(u) \in V(S_p)$ such that $b \leq_G u$, then $\mathcal{M}_{G,S}(V(G_b)) \cap V(S_p) = \emptyset$, this implies by Definition 2.2.1 that $lca_S(\mathcal{M}_{G,S}(V(G_b))) = lca_{S'}(\mathcal{M}_{G,S}(V(G_b)))$. □

Proposition 2.3.6. *[Subgraph of blue-nodes]*

Let S be a tree, G be a comparable tree, and $S' := SPR^*_S(p, y)$ for some $p, y \in V(S)$. Then, there exists a tree G^* such that $\mathsf{Rt}(G) \in V(G^*)$ and $V(G^*) = B$, where B is the set of blue-nodes in G.

Proof. We show that if $b \in V(G) \setminus \mathsf{Rt}(G)$ is a blue-node, then $\mathsf{Pa}_G(b)$ must also be a blue-node. This consequently results in a tree-subgraph G^* where each node in $V(G^*)$ is a blue-node and $\mathsf{Rt}(G) \in V(G^*)$.

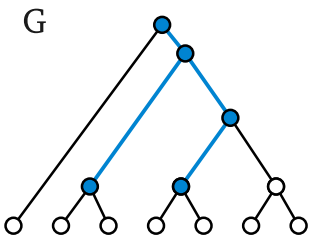


Figure 2.3.1 The blue nodes are forming a subtree within the gene tree G, indicating node mapping change.

Since b is a blue-node, by Lemma 2.3.5 there exists $\mathcal{M}_{G,S}(u) \in V(S_p)$ such that $b \leq_G u$, and there exists $\mathcal{M}_{G,S}(v) \in V(S) \setminus V(S_p)$ such that $b \leq_G v$. That implies $u, v \in V(G_b)$ and it follows that $u, v \in V\left(G_{\mathsf{Pa}_G(b)}\right)$. The operation $SPR^*_S(p, y)$ implies that either $y = \mathsf{Rt}(S)$ or $\mathsf{Pa}_S(p) = \mathsf{Rt}(S)$, therefore $\mathsf{Rt}(S') \notin V(S)$ and $\mathsf{Rt}(S) \notin V(S')$. Assume $y = \mathsf{Rt}(S)$; then $\mathcal{M}_{G,S'}(b) = \mathsf{Rt}(S')$, so $\mathcal{M}_{G,S'}(\mathsf{Pa}_G(b)) = \mathsf{Rt}(S')$, but $\mathsf{Rt}(S') \notin V(S)$. It follows that $\mathcal{M}_{G,S}(\mathsf{Pa}_G(b)) \neq \mathcal{M}_{G,S'}(\mathsf{Pa}_G(b))$, thus $\mathsf{Pa}_G(b)$ is a blue-node. Assume $\mathsf{Pa}_S(p) = \mathsf{Rt}(S)$; then $\mathcal{M}_{G,S}(b) = \mathsf{Rt}(S)$, so $\mathcal{M}_{G,S}(\mathsf{Pa}_G(b)) = \mathsf{Rt}(S)$, but $\mathsf{Rt}(S) \notin V(S')$. It follows that $\mathcal{M}_{G,S}(\mathsf{Pa}_G(b)) \neq \mathcal{M}_{G,S'}(\mathsf{Pa}_G(b))$, thus $\mathsf{Pa}_G(b)$ is a blue-node.

Therefore, $\mathsf{Pa}_G(b)$ must be a blue node. □

2.3.1.2 Structural properties of red nodes in species trees

Proposition 2.3.7. *[Red-node path]*

Let S be a tree, G be a comparable tree, and $S' := SPR^*_S(p, y)$ for some $p, y \in V(S)$. There exists the path $U := \{x | x \leq_S p\} \cup \{x | x \leq_S y\}$ in S that traverses all red-nodes. (see Figure 2.3.2).

Proof. U is a path that begins with the parent node p and ends at the regrafting position at edge $\left(y, \mathsf{Pa}_{\Psi(S)}(y)\right)$, because either $y = \mathsf{Rt}(S)$ or $\mathsf{Pa}_T(p) = \mathsf{Rt}(S)$.

Assume $\exists v \in V(S)$ such that v is a red-node and $v \notin U$, then there exists a blue-node $b \in V(G)$ such that $v = \mathcal{M}_{G,S}(b)$, then by Lemma 2.3.5 there exists $\mathcal{M}_{G,S}(u) \in V(S_p)$ such that $b \leq_G u$, but then $\mathcal{M}_{G,S}(b) \leq_S p$, so it follows that $\mathcal{M}_{G,S}(b) \in \{x | x \leq_S p\}$, thus $v \in U$. □

Corollary 2.3.8. *The path U is at most of length h nodes, where h is the height of S.*

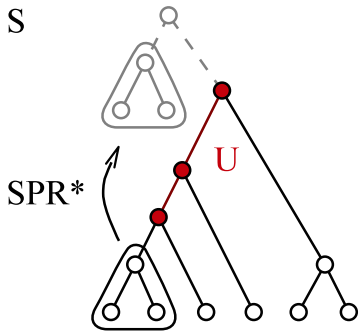


Figure 2.3.2 A SPR^*_S operation on a species tree where a subtree is regrafted to the root. The red-nodes form a path U in the species tree, along which mapping changes can occur.

2.3.2 Mapping Changes

Proposition 2.3.9. *[SPR mapping difference]*

Let S be a tree, G be a comparable tree, and $S' := SPR^*_S(p, y)$ for some $p, y \in V(S)$. Let B be the set of blue-nodes in tree S. Then, the mappings $\mathcal{M}_{G,S}\Delta\mathcal{M}_{G,S'}$ as defined in Problem 2.2.7 can be inferred by exclusively recomputing $\mathcal{M}_{G,S}(u)$ for all $u \in B$.

Proof. If $u \notin B$, then $\mathcal{M}_{G,S}(u) = \mathcal{M}_{G,S'}(u)$, thus the mapping $\mathcal{M}_{G,S'}(u)$ can be defined as follows:

$$\mathcal{M}_{G,S'}(u) := \begin{cases} \mathcal{M}_{G,S'}(u) & u \in B \\ \mathcal{M}_{G,S}(u) & \text{otherwise} \end{cases}$$

□

2.3.2.1 Algorithm to identify the set of blue nodes

Any node $b \in V(S)$ can easily be identified as blue by comparing the mapping $\mathcal{M}_{G,S}(b)$ and $\mathcal{M}_{G,S'}(b)$, i.e. b is blue if $\mathcal{M}_{G,S}(b) \neq \mathcal{M}_{G,S'}(b)$. Although, to compute $\mathcal{M}_{G,S'}(b)$ it may be necessary to compute mappings of descendants of b, or one can pre-compute $\mathcal{M}_{G,S'}$. The straight forward approach of identifying the set of blue nodes is to check every node $b \in V(S)$ for being blue or not. However, that would be as efficient as the straightforward solution for computing $\mathcal{M}_{G,S} \cap \mathcal{M}_{G,S'}$. Instead, we will compute a superset of blue nodes $\mathfrak{B}$.

Lemma 2.3.10. *[Superset of blue-nodes]*

Let S be a tree, G a comparable tree, and $S' := SPR^*_S(u,v)$ for some $u, v \in V(S)$. Then, Algorithm 2.1 computes the superset $\mathfrak{B}$ *such that* $B \subseteq \mathfrak{B}$, where B is the set of blue-nodes in $V(S)$.

Input:
A tree S and comparable tree G
A tree $S' \leftarrow SPR^*_S(u,v)$

Output:
$\mathfrak{B}$ // where $B \leftarrow \{g \in \mathfrak{B} | \mathcal{M}_{G,S}(b) \neq \mathcal{M}_{G,S'}(b)\}$ is the set of blue-nodes in $V(G)$

Algorithm:
$U \leftarrow \{x | x \leq_S p\} \cup \{x | x \leq_S y\}$ // The red-nodes in S
$M \leftarrow \{g_1, g_2, \ldots, g_{|V(G)|}\}$ such that $g_i \in V(G)$ and in pre-order traversal
for each $g \in M$ **do**
  if $\mathcal{M}_{T,S}(g) \in U$ **then** // (1)
    $\mathfrak{B} \leftarrow \mathfrak{B} \cup \{g\}$ // g is potentially a blue-node
  else
    skip subtree
  end if
end for
return $\mathfrak{B}$

Algorithm 2.1 Efficiently identifying the set of blue-nodes in a gene tree G when applying SPR^*_S on the species tree S.

Proof. First, we show that all blue-nodes in G are being traversed in Algorithm 2.1. All nodes in G are traversed except subtrees where step (1) fails, e.g. $\mathcal{M}_{T,S}(g) \notin U$. Let G_g be such a subtree, then $\mathcal{M}_{T,S}(g)$ is not a red-node, so g is not a blue-node. Assume $\exists$ a blue-node $b \in V(G_g)$, then $g \leq_G b$, so by Lemma 2.3.6 g is a blue-node which contradicts the assumption.

Next, we show that if $g \in M$ is a blue-node then $g \in \mathfrak{B}$, i.e. step (1) returns true. Let $v := \mathcal{M}_{G,S}(g)$, then v is a red-node by definition. The set U is constructed according to Lemma 2.3.7, so $v \in U$. Thus, step (1) identifies $g \in B$ correctly.

Therefore $B \subseteq \mathfrak{B}$ computes by Algorithm 2.1. □

Due to the fact that multiple gene duplications occur throughout the species tree, it is justified that the number of mappings to a species tree node can be bounded to an average in practice.

Proposition 2.3.11. *[Bounded complexity for $\mathfrak{B}$]*

Let S be a tree, G be a comparable tree, and $S' := SPR_S^*(p,y)$ for some $p,y \in V(S)$. Let $n := |V(S)|$, $m := |V(G)|$, and h be the height of S. Assuming the number of mappings to a species node is bounded by $O\left(\frac{m}{n}\right)$, then $\mathfrak{B}$ is of size $O\left(h\frac{m}{n}\right)$.

Proof. Let $s \in V(S)$ and $\mathcal{M}_{G,S}^{-1}(s) := \{b \in V(G) \,|\mathcal{M}_{G,S}(b) = s\}$ be the set of nodes mapping to s. If s is not a red-node, then the are no blue-nodes in $\mathcal{M}_{G,S}^{-1}(s)$, otherwise s is a red-node and by the given assumption $\left|\mathcal{M}_{G,S}^{-1}(s)\right| = O\left(\frac{m}{n}\right)$. From Corollary 2.3.8 it follows that there are at most $O(h)$ red-nodes in $V(S)$, thus $\mathfrak{B}$ is of size at most $O\left(h\frac{m}{n}\right)$. □

Complexity of Algorithm 2.1: The red-node path is computed in $O(n)$ time. The traversal through G is at most $2|\mathfrak{B}|$ nodes, so the traversal is bound to $O\left(h\frac{m}{n}\right)$ steps. Thus, Algorithm 2.1 time complexity is bounded by $O\left(n + h\frac{m}{n}\right)$.

2.4 Gene Duplication changes

In this section we introduce the gene duplication definition under the model by [37], which is computationally based on the LCA mapping. Then, we show how to efficiently compute the reconciliation score for sequential applied SPR operations on species trees as it is often used in search heuristics.

Definition 2.4.1. [Duplication]

Let S be a tree and G be a comparable tree. If a node $v \in V(G)$ is a (gene) duplication, then

$$Dup(v, \mathcal{M}_{G,S}) := \begin{cases} 1, & \mathcal{M}_{G,S}(v) \in \mathcal{M}_{G,S}(\mathsf{Ch}_S(v)) \\ 0, & \text{otherwise.} \end{cases}$$

Definition 2.4.2. [Reconciliation cost]

Let S be a tree and G be a comparable tree. $d(G,S) := \sum_{v \in V(G)} Dup(v, \mathcal{M}_{G,S})$ is the reconciliation cost from G to S. Let $\mathcal{G}$ be a set of trees comparable to S. $d(\mathcal{G},S) := \sum_{G \in \mathcal{G}} d(G,S)$ is the reconciliation cost from $\mathcal{G}$ to S.

Theorem 2.4.3. *Let $S' := SPR_S$, G be a comparable tree, and $L := \mathcal{M}_{G,S} \Delta \mathcal{M}_{G,S'}$. Then, $d(G,S') := d(G,S) + \sum_{v \in L} \left(Dup\left(v, \mathcal{M}_{G,S'}\right) - Dup\left(v, \mathcal{M}_{G,S}\right) \right)$.*

Proof. Let $K := V(G) \setminus L$. The theorem follows from:

$$
\begin{aligned}
& d\left(G,S'\right) - d\left(G,S\right) \\
&= \sum_{v \in V(G)} Dup\left(v, \mathcal{M}_{G,S'}\right) - \sum_{v \in V(G)} Dup\left(v, \mathcal{M}_{G,S}\right) \\
&= \sum_{v \in K} Dup\left(v, \mathcal{M}_{G,S'}\right) + \sum_{v \in L} Dup\left(v, \mathcal{M}_{G,S'}\right) - \sum_{v \in K} Dup\left(v, \mathcal{M}_{G,S}\right) - \sum_{v \in L} Dup\left(v, \mathcal{M}_{G,S}\right) \\
&= \sum_{v \in L} \left(Dup\left(v, \mathcal{M}_{G,S'}\right) - Dup\left(v, \mathcal{M}_{G,S}\right) \right)
\end{aligned}
$$

□

2.5 SPR local search heuristic

The SPR local search is a method often used in heuristics for large-scale phylogenetic analysis, when exact methods are computationally unwieldy. For instance, the gene-duplication problem is to find, for a given collection of gene trees, a comparable species tree with minimum reconciliation cost. The decision variant of the gene-duplication problem and some of its characterizations are NP-complete [46, 33] and it is not fixed parameter tractable in the number of gene duplications [9]. Therefore, in practice, heuristics (e.g., [54, 73, 8, 7]) are commonly used for the gene-duplication problem, even though they are unable to guarantee an optimal solution. These heuristics are based on local search and, consequently, involve repeatedly solving a local search problem.

These classical local search heuristics start with some species trees comparable with the input gene trees and finds a minimum reconciliation cost three in its neighborhood. This constitutes one local search step. The new tree thus found then becomes the starting point for the next local search step, and so on, until a local minima is reached. Thus, at each local search step, we must solve the local search problem. The time complexity of the local search problem depends on the tree edit operation used to define the neighborhood.

Our paper is of particular interest to heuristics where the local search is based on the SPR neighborhood. The SPR heuristic search is a stepwise process of optimizing a species tree,

where the SPR defines the local search neighborhood of species trees which are evaluated for a better reconciliation cost (lower gene duplication cost). The species trees that are evaluated in such a heuristic search can be traversed by sequentially applying SPR operations. By solving Problem 2.3.1 we efficiently obtain the difference in LCA mappings L for each species tree S in this traversal, from which we can quickly conclude the reconciliation score $d(\mathcal{G}, S)$ for a set of comparable trees $\mathcal{G}$.

Definition 2.5.1. [SPR neighborhood]

Let T be a tree, then the SPR neighborhood of T is $T := \{SPR_T(u,v) \,|\, u,v \in V(T) \text{ such that } SPR_T(u,v) \text{ is defined}\}$. For $n := V(T)$, the size of T is at least $\Theta(n^2)$.

Let $u \in V(T)$, then the partial SPR neighborhood of T is $\mathcal{SPR}(T,u) := \{SPR_T(u,v) \,|\, v \in V(T) \text{ such that } SPR_T(u,v) \text{ is defined}\}$. For $n := V(T)$, the size of $\mathcal{SPR}(T,u)$ is at least $\Theta(n)$.

Theorem 2.5.2. *All $\Theta(n^2)$ trees in T* can be traversed by sequentially applying at most $O(n^2)$ SPR operations.

Proof. Let $\{T_1, T_2, \cdots, T_k\} := T$. Then, for each T_i there exists $SPR_T(u_i, v_i)$ such that $T_i = SPR_T(u_{i1}, v_{i1})$, and $SPR_{T_i}(u_{i2}, v_{i2})$ such that $T = SPR_{T_i}(u_{i2}, v_{i2})$. Now, each tree in T is traversed in the following sequence of SPR operations: $SPR_T(u_{11}, v_{11}), SPR_{T_1}(v_{12}, u_{12}), \cdots, SPR_T(u_{k1}, v_{k1}), SPR_{T_k}(v_{k2}, u_{k2})$. This sequence is at most $2n^2$ long. □

Corollary 2.5.3. *Let S and S' be the trees in two consecutive local search steps. Then, there exists $S' = SPR_S(u,v)$, thus all species trees in a SPR heuristic search can be traversed by applying sequentially SPR* operations.

The classic local search solves Problem 2.5.4 in every local search step. We achieve a large speedup in practice if we relax this problem and let the local search solve Problem 2.5.5 instead. Then, the heuristic follows a search path that we describe as the *first descending path*. The classic heuristic search follows the steepest descending path.

Problem 2.5.4. [Steepest descending path SPR local search]

Instance: Let S be a tree and $\mathcal{G}$ be a set of comparable trees.

Problem: Find a tree S^* such that $d(\mathcal{G}, S^*) = \min_{T \in S} d(\mathcal{G}, T)$.

Problem 2.5.5. [First descending path SPR local search]

Instance: Let S be a tree and $\mathcal{G}$ be a set of comparable trees.

Problem: Find a tree $S' \in S$ such that $d(\mathcal{G}, S') < d(\mathcal{G}, S)$.

Algorithm 2.2 shows our first descending path heuristic. We designed this heuristic so it can make use of the $\Theta(n)$ amortized speed-up algorithm for the SPR local search [6] (applicable to step (1) in Algorithm 2.2). We also used this heuristic in our later experiments.

Input:
A tree S and a set of comparable trees $\mathcal{G}$
A $SPR_S^*(u, v)$ operation

Output:
A tree S' such that $d(\mathcal{G}, S') = \min_{T \in S'} d(\mathcal{G}, T)$

Algorithm:
Let Q be a round robin queue filled with $V(S)$ in random order.
while S has not changed for $|Q|$ times **do**
    $v \leftarrow$ next node in Q
    find a tree S' such that $d(\mathcal{G}, S') = \min_{T \in \mathcal{SPR}(S,v)} d(\mathcal{G}, T)$ (1)
    if S' was found **then**
        $S \leftarrow S'$
    end if
end while
return S

Algorithm 2.2 First descending path heuristic search.

2.6 Experiments

Empirical Data Sets – We first tested our algorithm and compared it against the heuristic implemented in DupTree [73] using three empirical data sets of very different sizes. First are gene trees from a published study of green plants (18896 gene trees; 136 total species; [18]). We also made sets of gene trees for the gymnosperm and Saxifragales plant clades. These

trees were made by downloading from GenBank (http://www.ncbi.nlm.nih.gov) the core nucleotide sequences from the specified clade and appropriate out-groups. The sequences were clustered into sets of homologs based on BLAST scores [2] and the clusters were aligned using MUSCLE [28]. The gene trees were inferred using ML implemented in RAxML [65] and rooted using the out-group taxa. The gymnosperm data set has 77 gene trees and 950 total species, and the Saxifragales data set has 51 gene trees and 958 total species. For each data set, we ran both our new heuristic and DupTree 100 times. All analyses were performed on an Intel® Xeon® 2.53GHz processor.

The fastest runs for the new heuristic ranged from 28x faster than DupTree in the green plant data set to over 200x faster in the gymnosperm data set (Table 2.1). Both heuristics appeared to quickly find an optimal topology for the green plant data set. In all the data sets except the green plants, the score of the best tree varied among runs, suggesting that the SPR search can get trapped in local optima. However, in 100 runs there was no evidence that our improved heuristic found worse trees than DupTree (Table 2.1). In fact, in the Saxifragales data set, the improved heuristic found trees with lower reconciliation cost than any trees found by DupTree (Table 2.1). The first descending path search is significantly faster than the descending steepest path search, and there is no evidence that the trees found were any less good.

	our algorithm				DupTree	
	first path		steepest path		steepest path	
	duplications	time	duplications	time	duplications	time
green plants	248757	31 sec	248757	3 min	248757	14 min
gymnosperms	1843 - 1880	7 sec	1843 - 1878	3.2 min	1847 - 1879	26 min
Saxifragales	813 - 848	4 sec	818 - 836	1.4 min	825 - 839	8 min

Table 2.1 Timings and final gene duplications (minimum - maximum of 100 individual searches) for the data sets green plants, gymnosperms, and Saxifragales. The local search heuristic follow "steepest path" and "first path".

Simulated Data Sets – We also performed analyses on data sets that we generated by simulation. We made both small data sets (400, 800, 1200, 1600, and 2000 species; Figure 2.6.1 and Figure 2.6.2) and large data sets (10,000, 20,000, 40,000, 60,000, 80,000, and 100,000 species; Figure 2.6.3). For both the small and large simulated data sets, we first generated a

species tree based on the Yule pure birth process using r8s [61]. For the small data sets, we then generated 500 gene trees with a minimum of 25 taxa and a maximum of half the number of taxa in the species tree, and for the large data sets we generated 1000 gene trees with a minimum of 5% of the total species and a maximum of 50% of the total species. To make the gene trees, we first randomly selected internal nodes in the species tree that had a sufficient number of descending leaves. We then randomly chose a number between 0 and 100%, and, if this percentage of leaves descending from our chosen node exceeded our minimum size requirement, we randomly selected this percentage of the descending leaves. We then made a subtree, our gene tree, which included only the randomly selected nodes. This process was repeated until we had the required number of gene trees. We finally introduced conflict to represent the conflict we observe in gene trees. The conflict consisted of SPR swaps on 25% of the nodes, moving them a maximum of 3 nodes from their original placement.

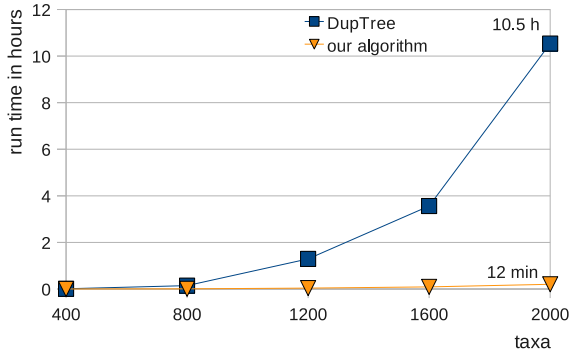


Figure 2.6.1 Performance comparison of heuristic searches on simulated data sets. The average run time of our algorithm (steepest descending path search) is with 12 minutes at its worst where DupTree (steepest descending path search) quickly reaches its limits.

We created 100 starting species trees with a simple leaf adding heuristic and proceeded with the best one, and then averaged over multiple heuristic searches. On the small data sets we ran both our heuristic and DupTree (Figure 2.6.1 and Figure 2.6.2). The 2000 taxa data sets took an average of 10.5 hours with DupTree, 12 minutes with our algorithm (steepest path), and only 18 seconds using first path and our algorithm. We found that larger data sets become computationally unwieldy with DupTree. Thus, we only ran the larger data sets with our new algorithm using first descending path, which was capable of inferring a species tree 50 times

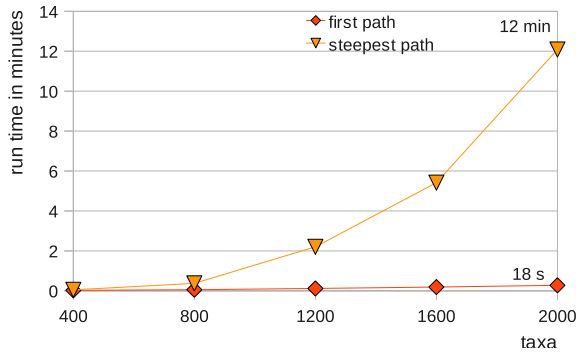


Figure 2.6.2 Performance comparison of heuristic searches of our algorithm on simulated data sets. The average run time of the first descending path search is with 18 seconds at its worst where the steepest descending path quickly reaches 12 minutes.

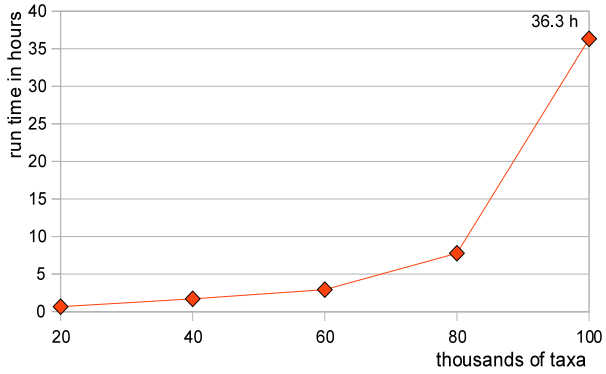


Figure 2.6.3 Average run time of our algorithm using the first descending path heuristic on large simulated data sets.

bigger than the small data set with 100,000 taxa in 36.3 hours on average.

2.7 Discussion

In this paper, we described two new approaches that vastly improve the speed of phylogenetic estimation based on GTP with little or no apparent loss in accuracy. In analyses of numerous empirical and simulated data sets, our new implementation of GTP is much faster than the GTP implementation in DupTree [73], which itself represented a tremendous increase in speed over the original naive SPR heuristics implemented in GeneTree [54]. With our new speedup it is now possible on a single desktop processor to use GTP to estimate species trees with 100,000 taxa, which, to our knowledge, is larger than any published phylogenetic trees

computed from scratch.

While faster heuristics are desirable, ideally the increase in speed will not result in finding less optimal solutions than slower heuristics. Reassuringly, in our analyses of numerous empirical and simulated data sets, the best reconciliation scores using the new, faster heuristics are very similar to, and sometimes better than, scores from the previous local SPR heuristic (Table 2.1, Figure 2.6.1). However, the SPR searches may get trapped in local optima. In this case, our new heuristic allows many more runs than previous heuristics, and this should lead to better results than relying on a single or few runs. For example, in our 2000 taxa simulated data set, we could run our new heuristic 2100 times while DupTree is performs a single SPR search. Still, we are currently working on new approaches, such as a ratchet search (see [50]), to ameliorate the problem of local optima.

Our new SPR heuristic is currently limited to the duplication cost model of gene tree species tree reconciliation. With incomplete gene sampling, it is difficult to distinguish a gene loss from missing data. Therefore, unless there is complete genomic sequence data for all taxa, which is very unlikely across huge species sets, it may be appropriate to count only gene duplications. Still, we note that any sequential SPR operation, like those used to infer trees under the duplication loss and deep coalescence cost models [4], can potentially be sped up with our initial algorithm, and we are working to generalize our heuristic speedup to other methods of gene tree reconciliation.

The new SPR heuristic suggests that gene tree reconciliation may be an effective method to infer the tree of life. A parallel computing approach was effective for increasing the speed and scale of previous GTP heuristics [76], and such an approach may enable our current approach to scale to the size of the tree of life. The GTP approach may be especially amenable to computing the tree of life. GTP uses gene trees as input, as opposed to large alignments, which greatly reduces the memory requirements and makes it practical to incorporate thousands of genes into an analysis. Furthermore, GTP based on duplications or duplications and losses do not require orthologous genes, and unlike ML or MP, GTP can easily incorporate data of large gene families in phylogenetic inference. With greater taxon sampling, it is more likely that gene trees will be incongruent due to events such as duplication, loss, deep coalescence,

or horizontal transfer, and thus, GTP may be especially appropriate for inferring large-scale phylogeny from multiple genes from multi-gene data sets. Still, we note that GTP is certainly not a replacement for more conventional phylogenetic methods like ML or MP. In fact, GTP relies on such methods to build the input gene trees, and the computational burden required to build the gene tree data sets likely far exceeds the computational burden of inferring the species tree using GTP. Thus, while our new heuristic approaches to GTP make it possible to estimate large-scale phylogenies from gene trees, estimating the tree of life will also require further speedup in other phylogenetic methods.

CHAPTER 3. A Scalable Parallelization of the Gene Duplication Problem

Modified from a paper published in the Journal of Parallel and Distributed Computing, Volume 70 Issue 3, March, 2010

André Wehe, Wen-Chieh Chang, Oliver Eulenstein, and Srinivas Aluru

Abstract

Phylogenetic is a branch of computational and evolutionary biology dealing with the inference of trees depicting evolutionary relationships among species and/or sequences. An important problem in phylogenetic is to find a species tree that is most parsimonious with a given set of gene trees, which are derived from sequencing multiple gene families from various subsets of species. The gene duplication problem is to compute a species tree that requires the minimum number of gene duplication events to reconciliate with the given set of gene trees. The best known heuristic algorithm for this NP-hard problem is a local optimization technique that runs in $O(n^2 + kmn)$ time per search step, where k is the number of gene trees, n is the size of the species tree, and m is the maximum size of a gene tree. In this paper, we present a parallel algorithm for the gene duplication problem that runs in $O\left(\frac{n^2+kmn}{p}\right)$ time for up to $p = O\left(\frac{nk}{\log k}\right)$ processors. Our algorithm exploits multiple levels of parallelism by parallelizing both the exploration of the search neighborhood and reconciliating of the gene trees with species trees in the neighborhood. Due to the wide variance in the sizes of the gene trees, it is difficult to completely characterize the behavior of the algorithm analytically. We present experimental results on the Blue Gene/L to study both levels of parallelism and how best they should be combined to achieve overall minimum execution time. On a large problem that took

about 62.5 hrs. on a 3 GHZ Pentium 4, our parallel algorithm ran in 7.7 minutes on a 1024 node Blue Gene/L.

3.1 Introduction

Large-scale phylogenetic analysis is of fundamental importance to comparative genomics and ubiquitous in evolutionary biology. Phylogenetics is the study of evolutionary relatedness among various groups of species. The rapid increase in available genomic sequence data provides an abundance of potential information for phylogenetic analysis. Most phylogenetic analyses combine genes from presumably *orthologous* loci, which are homologous sequences that were separated when a species diverged into two separate species. These analyses largely neglect the vast amounts of sequence data from gene families that are not orthologous, in which complex evolutionary processes such as gene duplication and loss, recombination, and horizontal transfer generate gene trees that differ from species trees. One approach to utilize the data from gene families in phylogenetics is to reconcile their gene trees with species trees based on an optimality criterion, such as the Gene Duplication model introduced by Goodman *et al.* [37]. This model includes *paralogous* genes, which are genes arising from duplication and not speciation. The corresponding optimization problem to the Gene Duplication model is called the *gene duplication problem* [39]. This problem is a type of *supertree problem*, that is, assembling from a set of input gene trees a species supertree that contains all species found in at least one of the input trees. The decision version of this problem is NP-complete [45]. Existing heuristics aimed at solving the gene duplication problem search the tree space of all possible species trees guided by a series of exact solutions to instances of a local search problem [54, 6, 7]. The gene duplication problem has shown much potential for building phylogenetic trees for snakes, vertebrates, *Drosophila*, and plants (e.g., [64, 55, 57]). Yet, the run time performance of existing heuristics has limited the size of such studies.

We improve on the fastest existing solution for the local search problem [6] by parallelization. Parallel methods have been developed for other important problems in phylogenetic analysis, for example maximum likelihood (RAxML [67]) and parsimony (Parallel Rec-DCM3 [27]) based phylogenetic reconstruction. However, to the best of our knowledge this contribution is the first

parallel solution to the Gene Duplication Problem. We achieve an asymptotic linear speedup for up to $O\left(\frac{nk}{\log k}\right)$ processors, where n is the number of nodes in the species tree and k is the number of gene trees. We demonstrate the scalability of our parallelization on a large number of processors and solve in minutes problems that have taken days of computational time on uni-processors. Using our method, we expect that even larger unsolved problem instances can be solved on larger scale parallel computers. We also compare our results with one of the best known sequential solutions [6], and argue that through parallelization we can perform a more thorough analysis and increase the quality of the result.

In the next section we review the gene duplication problem and the corresponding best sequential algorithm. In section III we describe and analyze the proposed parallelization. In section IV, we describe our parallel implementation and evaluate the experimental results. Section V concludes the paper.

3.2 Review of the Gene Duplication Problem

The gene duplication problem is based on the Gene Duplication model from Goodman *et al.* [37]. In the following, we (i) describe the Gene Duplication model, (ii) formulate the gene duplication problem, (iii) describe an efficient heuristic [54, 6] to solve the problem, and (iv) comment on important properties of the current best sequential solution [6].

Without evolutionary ambiguity a *species tree* and a *gene tree* are rooted, and fully binary trees that represent evolutionary relationships between species, and genes of a gene family, respectively. A gene tree and a species tree are *comparable*, if each leaf of the gene tree was sampled from a leaf of the species tree, i.e., the sampling can be described by a function called *leaf-mapping* that maps each leaf in a gene tree to a leaf in the species tree. An example of a *leaf-mapping* is shown in Fig. 3.2.1d

where the *leaf-mapping* is onto from the leaves a, b, c in gene tree G to the leaves A, B, C in species tree S.

The Gene Duplication (GD) model [53, 39, 49, 31, 80, 24, 16, 38] seeks to explain incompatibilities between a pair of compatible gene and species trees by reconciling the gene tree by invoking gene duplications and losses. For example, as shown in Fig. 3.2.1a-b, gene losses

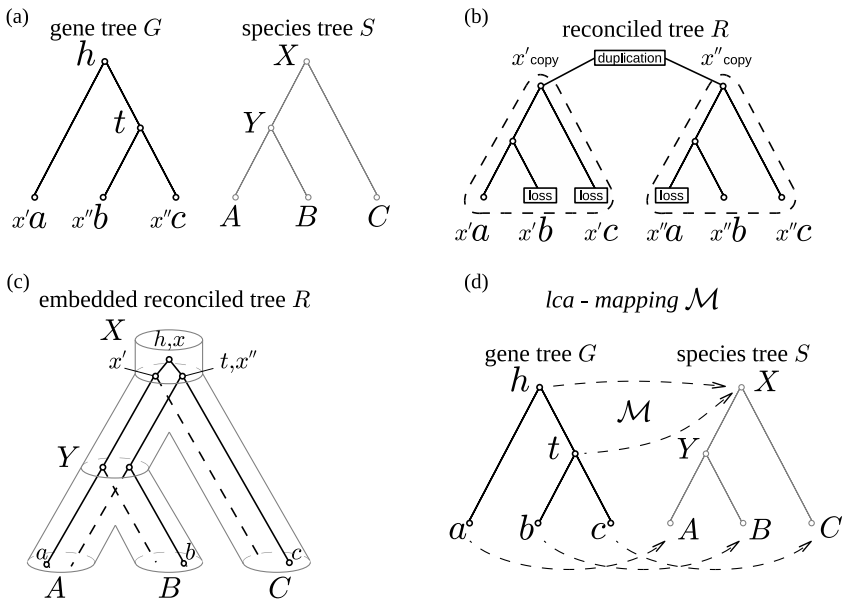


Figure 3.2.1 (a) An example of a rooted gene tree G and species tree S with disagreeing topologies. (b) R is the reconciled tree for G and S with one gene duplication (and 3 gene losses). The gene x duplicates into the genes x' and x''. (c) The solid lines in R represent the embedding of G into S with the gene duplication occurring in species X. (d) The GD model showing the *lca-mapping* and identifying the gene duplication in species X.

can prevent us from observing the complete set of genes, but a reconciled gene tree R can be inferred from the species tree S by duplicating a gene x in species X into the copies x' and x'', and letting both copies speciate according to the topology of S. In this case, as depicted in Fig. 3.2.1c, the gene tree can be embedded into the reconciled tree R, explaining the incompatibility by invoking the duplication of gene x.

Gene duplications that are invoked under the GD model to reconcile the gene tree are captured by extending the initial leaf-mapping to all nodes in the gene tree. This extended mapping, which is called *lca-mapping*, maps each gene to the most recent species that could have contained the gene. For every leaf in the gene tree this species is given by the leaf-mapping function. Every internal node g in the gene tree is mapped by the lca-mapping to the least common ancestor of the lca-mappings of its children. An example of an lca-mapping is shown in Fig. 3.2.1d. A gene in the gene tree represents a *duplication* event, if it maps to the same species as at least one of its children under lca-mapping. As an example in

Fig. 3.2.1, the lca-mapping maps gene h and its child t to the same species X. Therefore, x is a duplication. The reconciliation cost for a gene tree and a comparable species tree is the number of invoked duplication events required for reconciliation. For a given collection of gene trees, the reconciliation cost with respect to a given species tree is the overall sum of the individual reconciliation costs. The lca-mapping is linear time computable [80, 12, 41]. Hence, the reconciliation cost for a collection of gene trees and a species tree is computable in linear time.

Individual gene trees are inferred from phylogenetic analysis of sampled genes from their respective gene families. Given these, the *gene duplication problem* is to find a species tree with the minimum total reconciliation cost. The decision variant of this problem and some of its characterizations are NP-complete [45, 33], while some parametrizations are fixed parameter tractable [68, 40]. A standard local search heuristic used in GeneTree [52], subsequently made faster by a factor of $\Theta(n)$ by Bansal *et al.* [6], demonstrated that the gene duplication problem can be an effective approach for inferring high quality species phylogenies in practice. In this heuristic, a *tree graph*, representing the space of potential species trees, is defined for the given set of gene trees and some fixed tree edit operation. The nodes in the tree graph are the species trees and an edge is drawn between two nodes if the corresponding trees can be transformed into each other by a tree edit operation. The *reconciliation cost* of a node in the graph is the reconciliation cost of the species tree represented by that node and the given gene trees. Given a starting node in the tree graph, the heuristic is to find a maximal length path of steepest descent in the reconciliation cost of its nodes and to return the last node on such a path. This path is found by solving the local search problem for every node along the path. The *local search problem* is to find a node with the minimum reconciliation cost in the neighborhood of a given node. The time complexity of the local search problem depends on the tree edit operation used.

Let n denote the size of the species tree (number of nodes), m denote the maximum size of a gene tree, and k denote the number of gene trees. An edit operation of interest is the rooted subtree pruning and regrafting (SPR) operation [1, 17]. Given a tree S, an SPR operation can be performed in three steps (see Fig. 3.2.2): (i) prune some subtree P from S, (ii) add

a root edge to the remaining tree S, (iii) regraft P into an edge of the remaining tree S. The resulting tree graph is connected and every node has a degree of $\Theta(n^2)$, where n is the size of a species tree. The reconciliation cost for a species tree and the k gene trees can be computed in $O(n + km)$ time. Thus the local search problem for the SPR edit operation can be solved naively in $O(n^3 + kmn^2)$ time by evaluating separately each of the $\Theta(n^2)$ species trees in the neighborhood. However, it is possible to group the neighborhood into $\Theta(n)$ groups each containing $O(n)$ species trees such that the total reconciliation cost for all the trees in a group can be simultaneously evaluated in $O(n + km)$ time [6]. This reduces the total run-time of the local search problem to $O(n^2 + kmn)$. In this paper we present a parallelization of this algorithm and demonstrate that it allows us to solve problem sizes that were not possible with the sequential algorithm.

3.3 Solving the Gene Duplication Problem in Parallel

We propose and evaluate a parallelization of the sequential algorithm by Bansal *et al.* [6] that runs in $O(n^2+kmn)$ time. The algorithms starts with a given collection of k gene trees $\mathcal{G} := G_1, G_2, \ldots, G_k$ and a species tree S_0. Starting from S_0, the algorithm typically takes several hundred search steps to converge to a species tree that has no greater total reconciliation cost than any other tree in its neighborhood. Each search step consists of exploring the neighborhood of the current species tree S, obtained by performing a SPR tree edit operation on S. Each of the $\Theta(n^2)$ species trees in the neighborhood can be independently evaluated in parallel by computing its reconciliation cost with the gene trees. However this is wasteful, as the serial algorithm realizes synergies in evaluating multiple species trees by evaluating $O(n)$ species trees

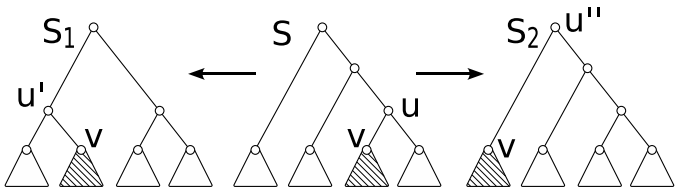


Figure 3.2.2 S_1 and S_2 are examples of trees in the SPR-neighborhood of S, obtained by pruning the subtree rooted at v and regrafting it into the remaining tree S.

with the same asymptotic cost as required for evaluating a single species tree. Our parallel algorithm conforms to this strategy and only parallelizes across the $\Theta(n)$ groups of species trees. We call this *Neighborhood Parallelization* (NHP). Within each group, reconciliation cost computation is needed for each gene tree versus the set of species trees within the group. As this task is independent for each gene tree, we parallelize across the set of gene trees. We call this *Gene Tree Parallelization* (GTP). It should be noted that the former is task parallelism and the latter can be classified as data parallelism. The two strategies can be independently applied; i.e., one can view the processor pool as partitioned into subsets of equal size with each partition computing the best choice among certain groups of species trees. Within each subset, each processor is allocated a set of gene trees for reconciliation computation. The species tree is replicated on all processors. This poses no memory issues as a tree with even a few million species can be easily accommodated, whereas trees that can be solved in practice are at most several thousand species.

The algorithm can be viewed as parallelizing across two dimensions — the neighborhood species trees and the k input gene trees. Correspondingly, the processors can be viewed as having two dimensional ranks (i, j), where $0 \leq i \leq p_1$, $0 \leq j \leq p_2$, and the total number of processors $p := p_1 \times p_2$. For processor with rank (i, j), i determines the species tree groups allocated to the processor and j determines the gene trees allocated to the processor. Due to the k-way parallelism at GTP level, and $\Theta(n)$-way parallelism at the NHP level, the combined parallelism available is quite large. An interesting issue is how to select the level of parallelization at each of these two levels for a given set of processors, so as to achieve good scalability and adjust the parallelization to suit different types of inputs.

3.3.1 Review of the Search Heuristic

In this section, we provide some of the details of the sequential algorithm necessary to understand our parallelization. For complete details, the reader is referred to [6]. The algorithm uses the SPR tree edit operation, which can be thought of as moving a subtree within a rooted species tree. More formally, the SPR operation for a tree S is defined as pruning a subtree S_v rooted at node v by cutting the edge connecting v to its parent u, and then regrafting the

subtree by the same cut edge in one of the following two ways:

1. Creating a new node u' which subdivides an edge in $S \backslash S_v$, and regrafting the subtree by the cut edge at node u'. Then, either suppressing the degree-two node u or, if u is the root of S, deleting u and the edge incident with u, making the other end-node of this edge the new root. In Fig. 3.2.2, tree S_1 is obtained from tree S using such an operation.

2. Creating a new root node u'' and a new edge between u'' and the original root. Then regrafting the subtree by the cut edge at node u'' and suppressing the degree-two node u. In Fig. 3.2.2, tree S_2 is obtained from tree S using such an operation.

The neighborhood of a species tree S is the number of different trees that can be obtained from S by an SPR operation. In an n-node tree S, $n-1$ subtrees can be selected for pruning, one corresponding to each edge. After pruning a subtree we have $O(n)$ possible edges to regraft into, the exact number depends on the pruned subtree. Each possible combination is a different SPR operation, resulting in the neighborhood size of $O(n^2)$. The objective is to compute the reconciliation cost for each of these species trees and select a tree with the lowest cost. The sequential algorithm simultaneously evaluates all species trees obtained from S by SPR operations that correspond to pruning the same subtree. To preserve the resulting sequential run-time savings, the parallel algorithm also does not distinguish between SPR operations resulting from the same subtree pruning for the purpose of distributing to processors. These will be allocated to a processor en massed as a block. Alternatively, NHP can be viewed as distributing subtrees to prune instead of individual SPR operations.

3.3.2 Parallelization on Gene Tree Level

Let S denote the current species tree and $\mathcal{G}$ denote the collection of input gene trees $G_1, G_2, \cdots, G_k$. Let $|G_i|$ denote the size of the gene tree G_i. The algorithm collectively evaluates all possible species trees in the neighborhood obtained by regrafting a subtree S' of S into $S \backslash S'$. The first step in the algorithm is to regraft the pruned subtree S' at the root by creating a new root and making $S \backslash S'$ and S' subtrees of the new root. The algorithm then pre-processes the resulting tree in $O(n)$ time to answer *lca* queries in constant time. Using this

information, the $O(n)$ species trees obtained by various possible regraftings of S' are collectively reconciliated with all the gene trees, and the minimum selected over these $O(n)$ species trees in the neighborhood. The corresponding serial run time is $O(n + km)$, where $m := max_{i=1}^{k}|G_i|$.

Let p_2 denote the number of processors used for gene tree parallelization. The k gene trees are distributed across processors while the species tree is present in each. The *lca* pre-processing has to be carried out serially on each processor. The reconciliation costs for each gene tree with respect to the $O(n)$ species trees are computed on the processor which is assigned the gene tree. For each of the species trees, a processor can compute the partial reconciliation cost due to the gene trees assigned to it. The total reconciliation cost is computed by a standard reduction operation involving all processors. This must be done for each of the $O(n)$ species trees as the tree with the minimum total reconciliation cost can only be identified after the total reconciliation cost is known for each of these species trees. Thus, the parallel run-time can be estimated as $O\left(n + \frac{k}{p_2}m + n\log p_2\right)$. Assuming $m = \Theta(n)$, which is observed in practice, the parallel run-time is simplified to $O\left(\frac{k}{p_2}n + n\log p_2\right)$ where the term $n\log p_2$ comes from the communication cost, and the serial run-time is $O(kn)$. Hence linear speedup is achieved when $n\log p_2 = O\left(\frac{nk}{p_2}\right)$, i.e., $p_2 = O\left(\frac{k}{\log k}\right)$.

For effective parallelization on the gene tree level, the sum of the gene tree sizes assigned to a processor should be balanced. The ideal case is when all gene trees have the same size. However, gene tree sizes can range from a few nodes to trees that are larger than the species tree. A great variance in sizes is also observed in practice. Thus, one cannot merely assign the same number of gene trees per processor but need to balance the total size allocated to a processor. We tested two different allocation strategies: In the first one, the gene trees are allocated to processors in a round robin fashion with no explicit attempt to do load balancing. This can work for well behaved inputs where the benefit of load balancing is outweighed by the cost of it. In the second strategy, the gene trees are first sorted based on their sizes. The trees are then allocated in decreasing size order with the next tree assigned to the least loaded processor. An additional point to note is that the gene trees remain invariant over all the steps in the search process. Thus, it is sufficient if the one time cost of load balancing is absorbed by its cumulative benefit over all search steps.

3.3.3 Parallelization Based on Tree Edit Operations

Independently of gene tree parallelization, the neighborhood of a species tree can be evaluated in parallel to find the tree with the smallest total reconciliation cost. As mentioned before, the $O(n)$ species trees in the neighborhood resulting from the pruning and regrafting of a single subtree are evaluated in a single block as their combined evaluation has the same asymptotic cost as each individual evaluation. However, the $n-1$ possible subtrees available for pruning create opportunities for neighborhood parallelization (NHP). The number of processor groups that can be utilized for NHP increases as a function of the species tree size.

Let p_1 denote the number of groups of processors used for NHP, where GTP is used within each group of p_2 processors as before (recall that the total number of processors p is $p_1 \times p_2$). For NHP, the subtrees of the species tree to be regrafted are distributed to different processor groups (each processor within a group is assigned the same subtrees but each processor works on computing reconciliation with its own set of gene trees). Assume a representative processor in each group has the minimum total reconciliation cost (and the corresponding tree) of the species trees assigned to its group. At the end, the minimum total reconciliation cost across all groups can be computed with a standard reduction operation. The parallel run-time can be estimated as $O\left(\frac{n}{p_1} \times T_{p_2,GTP} + \log p_1\right)$, where $T_{p_2,GTP} = O\left(\frac{k}{p_2}m + n \log p_2\right)$ is the parallel run-time for gene tree parallelization within each group of size p_2. As $\log p_1 = o\left(T_{p_2,GTP}\right)$, the parallel run-time is simplified to $O\left(\frac{n}{p_1} \times T_{p_2,GTP}\right)$, where linear speedup is obtained for $p_1 = O(n)$. When NHP and GTP are effectively combined, up to $O\left(\frac{nk}{\log k}\right)$ processors can be used to achieve linear speedup.

The $n-1$ available subtrees for pruning do not each have the same number of possibilities for regrafting. By taking the number of regraft positions as a measure of the cost of computing this portion of the neighborhood, one can attempt to assign the subtrees to processor groups in a load balanced fashion. Once again, we tested the round robin scheduling strategy with no explicit load balancing, and a load balancing strategy based on the sorted order of the number of regraft positions. In contrast to gene tree parallelization, the species tree changes each iteration. Thus, the load balancing needs to be computed in each search step afresh.

3.4 Experimental Evaluation and Results

We developed a parallel software that solves the gene duplication problem incorporating both NHP and GTP parallelization and switches to activate load balancing or simple round robin scheduling. We use a simple hill climbing heuristic for all of our experiments. Our implementation is in C++ and MPI and is built from components of the corresponding sequential program [6] and the C++ Standard Template Library. The memory consumption is generally very low (less than 10 MB/process for our experiments) even for larger problem sizes, indicating the compute-intensive nature of the problem. The inter-processor communication of our implementation is exclusively based on the collective MPI-routines `MPI_Allreduce` and `MPI_Reduce`, which are often optimized for the underlying hardware architecture.

We carried out an experimental evaluation on the IBM Blue Gene/L to (i) demonstrate the scalability of the parallel algorithms, (ii) determine the effect of NHP and GTP parallelization using multiple data sets both synthetic and derived from biological sequence data, and (iii) study the optimum use of hybrid parallelization for a given set of processors. In addition, we investigated the co-processor and virtual node modes of the Blue Gene/L to determine their impact on execution time. Our experimental platform is a 1,024 node system, with two processors and 512 MB of memory per node.

We represent the data sets in the format $n' \times k$, where n' is the number of leaf nodes in the species tree (also known as the number of taxa), and k is the number of gene trees in the input. The size of a species tree is given by $n = 2n' - 1$. While theoretically the size of a gene tree can be unbounded, in practice, its size is $O(n)$. Hence, the work per search time step, which is $O(n^2 + kmn)$, behaves as $O(kn^2) = O(kn'^2)$. While we let gene trees assume their natural sizes when these trees are inferred from biological data sets, we maintain their sizes to be $O(n)$ when creating purely synthetic data sets to be close to what is observed in real data sets. Throughout the experiments, p denotes the total number of processors, and p_1 and p_2 denote the extent of the NHP and GTP parallelization, respectively (recall, p is $p_1 \times p_2$). We study their effectiveness independently (by setting p_1 as p, or p_2 as p), and also in combination. For combined parallelism, we provide p and p_1, leaving p_2 to be inferred.

3.4.1 The Empirical Data Sets

Two empirical data sets of sizes 1061×126 and 624×3978 are used to evaluate the performance of the parallel algorithms. They are both built from green plant gene family sequences available in GenBank (http://www.ncbi.nlm.nigh.gov) following a common phylogeny construction procedure [6]. These two sets are used as one has a relatively large species tree and the other has a relatively large number of gene trees.

The run-times are reported for the total solution of the problem instead of time per search step. Note that multiple trees in the local neighborhood of a species tree can have the same total reconciliation cost, which might cause ties for selecting the species tree for the next iteration. While ties can be broken arbitrarily, each selection can potentially lead to a different search path with different number of steps. For consistency, we enforce the same selection across different runs while varying the number of processors and the extent of NHP/GTP parallelization. Finally, we also explore different starting points for the search to gauge the variation on path lengths to reach local optimum.

3.4.2 Run-time and Scalability on Empirical Data Sets

In the first series of tests, we evaluated NHP and GTP independently for varying numbers of processors by setting p_1 as p to study NHP and p_2 as p to study GTP. The run-time in seconds for the two empirical data sets as p is varied from 4 to 1024 are shown in Table 3.1. Note that the maximum number of processors that can be used for NHP is $n = 2n' - 1$ and the maximum number of processors that can be used for GTP is k. The unfilled entries in the upper portion of the table indicate that the problem size was too big to run on configurations with few processors due to the long run times it would entail (memory is no limiting factor for this problem). Although these runs are definitely doable, their collective execution time runs into several days, and the insight to be gained here is very little as we are operating well under the scaling limits.

Recall that the maximum numbers of processors that can be utilized while obtaining linear scaling are $O(n)$ for NHP and $O\left(\frac{k}{\log k}\right)$ for GTP, respectively. While there is bound to be

Table 3.1 Run-time in seconds of NHP and GTP parallelization on empirical data sets of sizes 1061×126 and 624×3978. Note that GTP cannot use more processors than the number of gene trees.

	p	4	8	16	32	64	126	128	256	512	1024
1061×126	NHP	21,690	12,073	6,394	3,384	1,755	–	879	443	196	109
	GTP	19,626	10,853	6,840	4,620	3,496	2,929	–	–	–	–

	p	64	128	256	384	512	768	1024
624×3978	NHP	16,830	8,880	4,000	2,478	1,872	1,257	1,241
	GTP	12,126	6,752	3,963	3,070	2,508	1,961	1,751

some variability in practice because the theoretical predictions assume certain simplifications (such as not taking into account the significant variance in gene tree sizes), it is interesting to see if the various observations indicate the same or a small range of constant factors that provide agreement with the theoretical predictions. From the table, NHP attains near perfect scaling for roughly as many as n' or $\frac{1}{2}n$ processors, indicating that just two pruned subtrees are sufficient to provide adequate balancing of the load. The reason behind this relatively large degree of parallelism provided by NHP is as follows: When a subtree of S' is pruned and its regrafting into every possible edge of $S \backslash S'$ is considered, the first part of the computation is the *lca* pre-processing of the tree obtained by regrafting S' at the root of $S \backslash S'$. As the size of the resulting tree is always n, this work remains the same independent of the size of S' and $S \backslash S'$. In the next phase, the computation depends on the number of positions to regraft, or the size of $S \backslash S'$. Thus, the change in load attributable to the different choices of the pruned subtree S' varies within a small constant factor. This provides good scaling when distributing the neighborhood for parallelization. On the 1061×126 data set, NHP provides a speedup of 853 over 1024 processors when compared with the run-time of the sequential algorithm (not the parallel algorithm running on one processor). See Fig. 3.4.1.

The gene trees have considerable variability in their sizes and multiple trees are needed per processor to balance the load. Since only $O\left(k/\log k\right)$ processors can be used for linear scaling, this automatically requires $\Omega(\log k)$ trees per processor. Once again, the data in Table 3.1 confirms that the maximum number of processors that can be used while preserving linear scaling closely tracks the $O\left(k/\log k\right)$ limit with a small constant factor of about $\frac{3}{4}$. While only

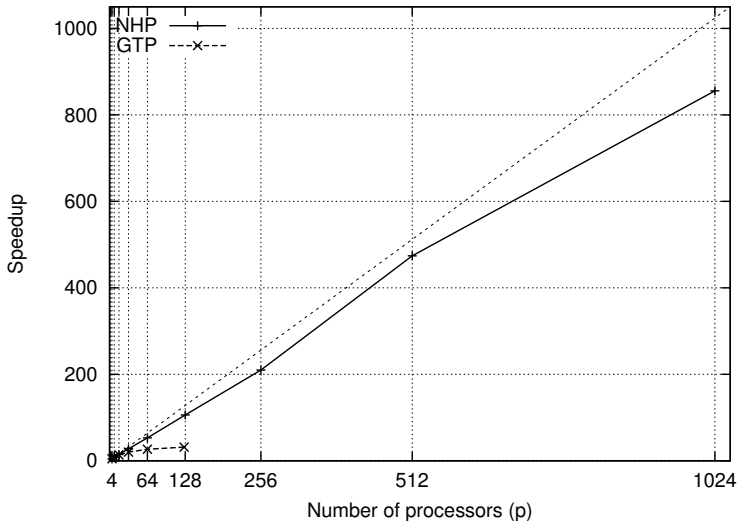


Figure 3.4.1 Speedup for NHP and GTP as a function of number of processors, when compared to the sequential algorithm for the empirical data set of size 1061×126. NHP provides impressive scaling up to 1024 processors. The GTP graph is hard to assess from the figure but it can use no more than 126 processors due to the data size, and provides perfect scaling up to 16 processors, in line with the $O(k/\log k)$ behavior.

two data sets are shown here, these predictions are confirmed in several other experiments.

3.4.3 Optimal Combination of NHP and GTP

The second series of tests conducted is to study various combinations of NHP and GTP parallelization for a given number of processors, and determine their optimal combination. Each graph in Fig. 3.4.2 corresponds to a fixed p. The graph is obtained by varying the number of NHP groups (p_1). An additional graph is drawn to show the optimal combination as a function of p. This is obtained by choosing the lowest point from each of the other graphs. Several observations are in order: effective combination of NHP and GTP provides perfect scaling on up to the 1024 processors we could test on, and should easily extend beyond to larger number of processors. As NHP parallelization provides linear speedup until 512 processors, the optimum for up to 512 processors is obtained by NHP parallelization itself, while the 1024 processor optimum includes 2-way GTP. More such tests, using synthetic data sets, indicate that it is good to use at least some degree of GTP as well, as it exhibits excellent scaling at least initially.

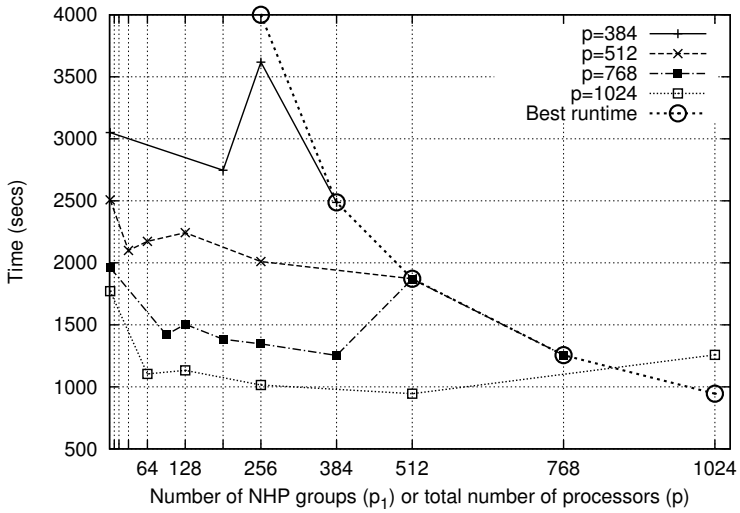


Figure 3.4.2 Four of the graphs in the figure show run-time as a function of the number of NHP processor groups (p_1) for $p = 384$, 512, 768, and 1024. The additional graph depicts best run-time as a function of the total number of processors, and is obtained from the other four. Note that the bumps in the curves for the p valyes of 384 and 768 are most likely caused by uneven division between GTP and NHP groups (e.g., $p = 384$ while $p_1 = 256$, and $p = 768$ while $p_1 = 512$, resulting in $p_2 \in \{1, 2\}$).

3.4.4 Synthetic Data Sets

To carry out further tests, we generated synthetic data sets using the following procedure: A gene tree is at first a random tree of n' taxa, and then it is reduced to m' taxa randomly, where m' has a normal distribution between 3 and n'. This distribution is an attempt to model the empirical data sets based on our observations. The data set consists of k such gene trees, and an initial species tree chosen at random from the union of gene tree taxa sets. Although it is possible that a taxon can be removed coincidentally from all gene trees, thus disappearing from the species tree, we did not encounter such a case while generating the data sets because k is sufficiently large.

The run-time performance for three data sets, each containing 128 gene trees and varying number of species trees, under NHP is shown in Fig. 3.4.3. We observed that linear speedup is achieved with up to $\frac{1}{4}n$ to $\frac{1}{2}n$ processors. Similarly, GTP performance for three data sets with 128 taxa in the species tree, while varying the number of gene trees, is shown in Fig. 3.4.4.

When the run-time is normalized by dividing with the number of search steps, the results show that the work scales quadratically with n and linearly with k, as expected.

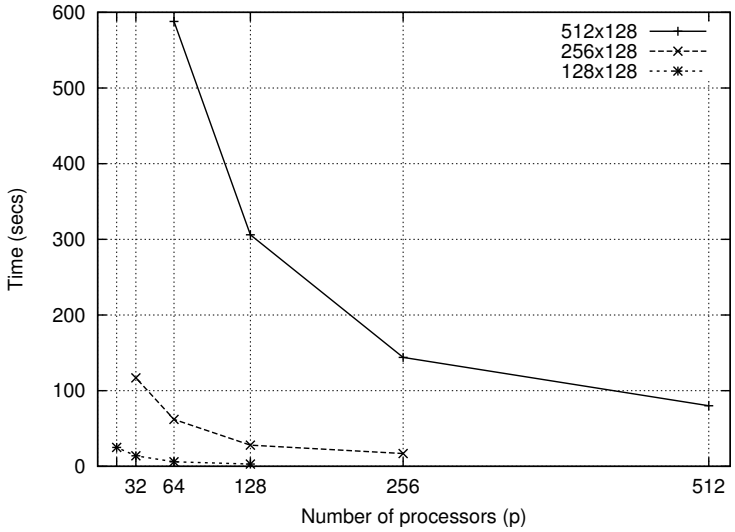


Figure 3.4.3 Performance of NHP on simulated data sets. The number of iterations for reaching the local optima are 412, 192, and 85 for the 512×128, 256×128, and 128×128 data sets, respectively.

The graphs in Fig. 3.4.5 depict the run-time on 1024 processors using various combinations of NHP and GTP parallelism. Each of the graphs corresponds to a different problem size. The lowest point in each graph indicates the optimal combination of NHP and GTP. Note that n' sizes are chosen to be 513, 257 etc., so that $n = 2n' - 1$ is just above 1024, 512 etc., allowing us to conduct tests up to the maximum degree of parallelization. In most cases, optimal performance is obtained by using $p_1 := \frac{n}{4}$, i.e., selecting $\frac{n}{4}$ NHP groups and using GTP parallelization for the rest.

Thus, the strategy for determining the extent of NHP and GTP parallelism for a given number of processors can be summarized as follows: The total number of processors is bounded by $O\left(\frac{nk}{\log k}\right)$. The number of NHP processor groups can be chosen as high as $\frac{1}{4}n$. Note that while this is a conservative estimate representing the lowest observation among the synthetic and empirical data sets we tested, it ensures that linear scaling is obtained, assuming the range of tests we conducted are a good representative. The rest, if needed, should come from GTP parallelization. However, if p is within the limits of largest number of permissible NHP groups

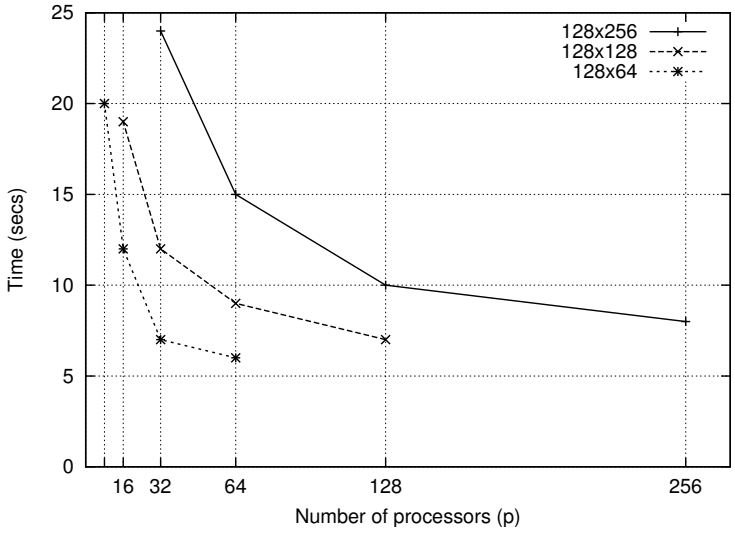


Figure 3.4.4 Performance of GTP on simulated data sets. The number of iterations for reaching the local optima are 101, 85, 79 for the 128×256, 128×128, and 128×64 data sets, respectively.

($\frac{1}{4}n$), it is still good to use a modest 2- or 4-way GTP parallelism due to its excellent initial scaling.

3.4.5 Load Balancing and Virtual Node Mode

The next series of tests we conducted are to study the effect of load balancing on execution time. We tested two scheduling strategies for both NHP and GTP. The first is a simple round-robin scheduling strategy where trees are assigned in a round-robin fashion to the processors. The second is a strategy to balance the load by sorting the trees according to their sizes, and allocating them in that order so that the next tree is assigned to the processor that currently has the least load. We also studied use of the Blue Gene/L in coprocessor and virtual node modes. In the former, one MPI process is run on a node with a processor handling computation and the other handling communication. In the latter, both processors are used to run two different threads, thus doubling the available parallelism. The results for combinations of both scheduling strategies and coprocessor/virtual node mode are shown in Table 3.2.

With respect to studying the effect of load balancing, we present run-time for four alternatives: (i) no load balancing (i.e. round-robin), (ii) load balancing only for GTP, (iii) load

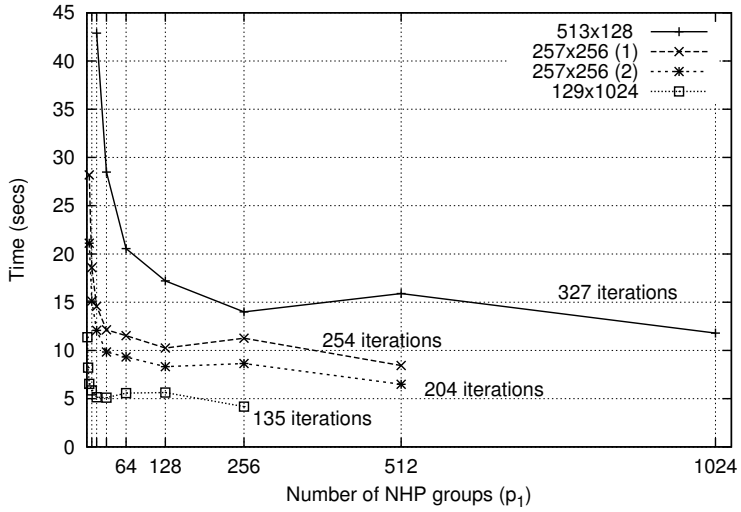


Figure 3.4.5 Performance curves of simulated hybrid approach.

Table 3.2 Run-time with and without explicit load balancing and the effect of coprocessor and virtual node modes. All runs are on 1024 processors and p_1 denotes the number of NHP groups. Column headers indicate number of processors, which is the number of nodes in coprocessor mode and twice the number of nodes in virtual node mode.

$p = 1024$	coprocessor				virtual-node			
p_1	1	32	512	1,024	1	32	512	1,024
Round-robin	1,899	779	921	1,224	1,911	795	946	1,252
GTP balancing	1,754	781	939	1,240	1,843	800	961	1,274
NHP balancing	1,879	1,043	923	1,226	1,911	1,063	959	1,251
NHP + GTP balancing	1,751	1,042	935	1,241	1,844	1,063	967	1,275

balancing only for NHP, and (iv) load balancing for both GTP and NHP. In most cases, the cost of load balancing outweighs its benefits, even for gene tree parallelization where the cost of load balancing can be amortized over all iterations. The only exception is when only GTP is used (corresponding to $p_1 = 1$). Even in this case, GTP load balancing alone provides as good a result as load balancing at both GTP and NHP. Thus, NHP load balancing is never beneficial, except for negligible incremental gains occasionally, as seen in Table 3.2.

Another conclusion supported by uniform evidence is that the coprocessor mode provides marginal benefits ($< 3\%$) over the virtual node mode. Note that the same number of MPI threads are compared in Table 3.2; i.e., $1,024$ processors in the coprocessor mode corresponds

to 1024 nodes, while 1024 processors in the virtual-node mode corresponds to 512 nodes with both processors utilized for running MPI threads. Thus, it is better to use the given machine resource in the virtual node mode so that twice as many processors can be utilized obtaining nearly a factor of 2x over running in coprocessor mode. This is not surprising as the only communication steps needed in the algorithm are a reduction within each NHP group after reconciliation of gene trees with species trees corresponding to the pruned subtrees assigned to the group, and another reduction involving a representative from each NHP group to compute the tree with minimum total reconciliation cost. The algorithm is compute intensive, and uses communication sparingly to collect results, thus wasting resources if a coprocessor is designated to handle communication costs.

3.4.6 Multiple Explorations of Search Space

The significant run-time advances made possible by our parallel algorithm can be also be used to increase the quality of the solution. Note that on the largest problem studied, we reduced the run-time from over two and a half days of compute time on a standard workstation to about 7.7 minutes on 1024 nodes. Taking advantage of this, one can carry out multiple explorations of the search space by carrying out the local search heuristic with many starting points instead of just one. It is tempting to use this as a third level of parallelization and first partition the processors into as many groups as the number of independent runs prior to applying our program and exploit NHP + GTP parallelization. However, it should be kept in mind that while the time per search step will be uniform across the multiple searches, the number of search steps to convergence can vary significantly. For example, some of tests we conducted resulted in search steps ranging from the low hundred to several hundred search steps. Thus, if NHP + GTP can utilize all available processors effectively, it is still preferable to carry out the searches one after another. If the available parallelism is much greater, such as in forthcoming petaflop computers, one could partition processors into groups with each group carrying out exploration from one random starting point after another, until a pre-specified execution time is reached and the best solution thus far recorded.

3.5 Conclusions

In this paper, we present a parallel algorithm for the gene duplication problem, an important problem that arises in phylogenetics. Using two levels of parallelism that can be utilized independently, we developed a parallel framework that reduces run-time from days to minutes on practical data sets. We carried out a thorough experimental evaluation of the algorithm and present strategies for optimum utilization of a specified number of processors. Our long term goal is to build high quality phylogenetic trees and solve large-scale phylogeny problems using the computational capability afforded by parallel computers.

Acknowledgments

We wish to thank Gordon Burleigh for providing the empirical data sets used in our experimental analysis. This research is supported in part by National Science Foundation under CNS-0521568 and CCF-0431140.

CHAPTER 4. Algorithms for Knowledge-Enhanced Supertrees

Modified from a paper accepted by the International Symposium on Bioinformatics Research and Applications 2012

André Wehe, J. Gordon Burleigh, and Oliver Eulenstein

Abstract

Supertree algorithms combine smaller phylogenetic trees into a single, comprehensive phylogeny, or supertree. Most supertree problems are NP-hard, and often heuristics identify supertrees with anomalous or unwanted relationships. We introduce knowledge-enhanced supertree problems, which seek an optimal supertree for a collection of input trees that can only be assembled from a set of given, possibly incompatible, phylogenetic relationships. For these problems we introduce efficient algorithms that, in a special setting, also provide exact solutions for the original supertree problems. We describe our algorithms and verify their performance based on the Robinson Foulds (RF) supertree problem. We demonstrate that our algorithms (i) can significantly improve upon estimates of existing RF-heuristics, and (ii) can compute exact RF supertrees with up to 17 taxa.

4.1 Introduction

There is growing interest in using phylogenetic trees to investigate evolutionary and ecological processes, including the maintenance of biodiversity [35, 25] and the effects of global change [29, 78, 72]. These analyses often require extremely large trees that include species from many disparate clades. Most traditional phylogenetic analyses build trees for either relatively

few closely related taxa or exemplar taxa from distantly related clades. Thus, there is a need for methods that can synthesize the existing phylogenetic data. Supertree problems, which take input trees with partially overlapping taxon sets and seek a tree (the supertree) containing all of the taxa in the input trees, provide one such approach (e.g., [62, 13, 14]).

Most standard supertree problems are NP-complete [10, 46, 81]. Like other NP-complete problems in general, exact solutions can be obtained by exhaustive search. However, run-times typically become prohibitive already for very small instances. Heuristic searches, often based on local search algorithms, are necessary to estimate solutions for larger data sets. While heuristic solutions appear to work well in many cases, one persistent criticism of supertree methods is that the resulting supertrees often contain relationships that appear to be anomalous, or at least unwanted. In some cases, this may result from the failure of a supertree heuristic to find an optimal solution [5, 23]. In other cases, these relationships may be a symptom of undesirable properties of the supertree problem. One approach to avoid problematic relationships and to improve the efficiency of heuristic searches is to provide a set of possible phylogenetic relationships from which the supertree has to be assembled. These phylogenetic relationships might be provided by existing supertrees or other phylogenetic hypotheses, and can be incompatible.. Focusing the supertree search on these candidate relationships would not only prevent the inclusion of undesired relationships in the supertrees, which are excluded from the set of candidates, but it also can tremendously reduce the search space for supertree solutions.

Here we introduce knowledge-enhanced supertree problems that construct supertrees from credible phylogenetic knowledge. Given a collection of input trees and a set of possibly contradictory phylogenetic relationships, a knowledge-enhanced supertree problem seeks an optimal supertree only in the tree space described by the phylogenetic relationships. We provide efficient algorithms for these problems that, in a special setting, can compute exact solutions for the original supertree problems. We focus on the RF supertree problem to describe our algorithms and to verify their performance in practice. We demonstrate that our knowledge-enhanced algorithms improve upon RF supertrees estimated by the heuristic of Bansal *et al.* [5]. Further, we show that our algorithms can compute, for the first time, exact RF supertrees for up to 17 taxa within 9 hours.

Related work. Knowledge-enhanced searches have been an effective approach to guide local search heuristics for supertree, and other phylogenetic, problems. In these approaches, unresolved trees or clades are passed as constraints to the heuristic's local search method (e.g., RAxML [66], PAUP [69], and DupTree [73]), which forces the solution supertree to agree with the constraints. Although such constraints can be effective, they may be inefficient. The constraint search must consider all possible topologies consistent with the constraints, rather than a specific set of possible incompatible topologies. Furthermore, these constrained approaches are still heuristics and therefore risking suboptimal solutions.

In this work, we introduce knowledge enhanced supertree problems and provide such a definition based on the Robinson-Foulds (RF) supertree problem. The RF supertree problem seeks a binary supertree that minimizes the sum of the RF distances between every input tree and the supertree. This problem is NP-hard [10], and therefore, it has been addressed by local search heuristics [5, 23]. These heuristics allow for time efficient estimation of RF supertrees for data sets with hundreds of taxa, but experiments suggest that they may become stuck at locally optimal solutions [5, 23]. The RF supertree problem can also be solved exactly with exhaustive search, but this becomes prohibitively time consuming for already small instances (e.g., approximately 8 taxa). Exact algorithms that are substantially faster than exhaustive search have been developed for several other supertree problems [79, 71, 21]. However, no such algorithms have been described for the RF supertree problem.

Our Contribution. We define knowledge-enhanced RF supertree problems, and provide efficient algorithms to solve them. *Knowledge-enhanced RF supertree problems* seek an optimal RF supertree for a given collection of input trees that can only be assembled from a given set of, possibly contradictory, phylogenetic relationships. These relationships can be described either as clusters or sibling-clusters. *Sibling-clusters* are two disjoint clusters whose parent in a rooted phylogenetic tree is the union of these clusters. Such phylogenetic relationships are available in abundance, for example, as previously inferred phylogenetic estimates, including estimates from supertree heuristics. We describe efficient algorithms that solve the knowledge-enhanced RF supertree problems for n taxa and m nodes of the input tree for either c clusters or t siblings clusters in time $O\left(c^2nm\right)$ and $O\left(tnm\right)$ respectively. We verify the performance of

these algorithms for large empirical collections of trees. Our algorithms significantly improve on estimates of these collections that were computed by the RF heuristic from Bansal *et al.* [5].

We also describe an exact algorithm for the RF supertree problem that is a special version of the knowledge-enhanced algorithms. Our exact algorithm improves on the best known (naïve) solution for the RF supertree problem by a factor of $\sqrt{n^n}/3^n$, where n is the number of taxa in the supertree. This substantial improvement allows, for the first time, to compute exact RF supertree with up to 17 taxa.

Finally, we show how our new algorithmic approaches for the RF supertree problem lead to efficient algorithms for the knowledge-enhanced versions of the following supertree problem: the duplication and loss problem, the deep coalescence problem, and the MRP problem. The exact algorithms for the original supertree problems are a special setting of the corresponding knowledge-enhanced algorithms.

4.2 Basic Notation and Preliminaries

In the interest of brevity some proofs have been omitted, but are included in a technical report that is available on request from the authors. Let T be a rooted tree. We denote the vertex set, edge set, and leaf set of T by $V(T)$, $E(T)$, and $\mathsf{Le}(T)$ respectively. The root of T is denoted by $\mathsf{Rt}(T)$. Given a vertex $v \in V(T)$, we denote the children of v by $\mathsf{Ch}_T(v)$, and the parent of v by $\mathsf{Pa}_T(v)$. Two vertices in T are called *siblings* (of each other) if they have the same parent. We write (u, v) to denote the edge $\{u, v\} \in E(T)$ where $u = \mathsf{Pa}_T(v)$. The set of *internal vertices* of T, denoted $I(T)$, is defined to be the set $V(T) \setminus \mathsf{Le}(T)$. We call T *full binary* if each vertex $v \in I(T)$ has exactly two children.

Given a vertex-set $U \subseteq V(T)$, we denote by $T(U)$ the unique subtree of T that spans U with the minimum number of vertices. Furthermore, the *restriction* of T to U, denoted by $T_{|U}$, is the tree that is obtained from $T(U)$ by suppressing all non-root vertices of degree two. The *subtree* of T *rooted at* $v \in V(T)$, denoted by T_v, is defined to be $T_{|U}$, for $U := \{u \in \mathsf{Le}(T) \mid v \text{ is on the shortest path between } \mathsf{Rt}(T) \text{ and } u\}$.

Given a vertex $v \in V(T)$ we define the *cluster of* v to be $\mathcal{C}_T(v) := \mathsf{Le}(T_v)$, and the *cluster presentation* of T is defined as $\mathcal{C}(T) := \{\mathcal{C}_T(v) \mid v \in V(T)\}$.

Let X be a label set. A tree T is called an *X-tree* if $\mathsf{Le}(T) = X$, and called a *partial X-tree* if $\mathsf{Le}(T) \subseteq X$. The set of all X-trees is denoted by $\mathcal{T}(X)$, and the set of all X-trees that are full binary is denoted by $\mathcal{B}(X)$.

The following conventions are used for convenience throughout the manuscript. We write the set-operation $\cup$ as $\dot{\cup}$ when the corresponding intersecting sets are disjoint. Unless noted otherwise, the term tree refers to a rooted tree that has no vertices of degree two other than its root.

The definition for the *Robinson-Foulds (RF) distance* [60] is expressed as the dissimilarity of a pair of trees on the same taxon set. For rooted trees this dissimilarity is based on clusters. For the RF supertree problem the input trees are partial *X-tree*s, so trees may share only some of the taxa with the supertree. In this case the minus RF distance can be computed (e.g., [5, 23]), which is the *RF* distance on a pair of trees restricted to their shared taxon set as follows:

Definition 4.2.1. [minus RF distance]

Let S be an X-tree, and T be a Y-tree where $Y \subseteq X$. We define $RF(T,S) := \left|\mathcal{C}(T) \,\Delta\, \mathcal{C}\left(S_{|Y}\right)\right|$, where Δ denotes the symmetric difference. Given a set P of partial X-trees, we define the minus *RF distance from P to S* as $RF(P,S) := \Sigma_{T \in P} RF(T,S)$, and the minus *RF distance of P* in the scope of $\mathcal{B}(X)$ as $RF(P) := \min_{S \in \mathcal{B}(X)} RF(P,S)$.

Problem 4.2.2. [RF supertree (NP-hard [10])]

Instance: A set P of partial X-trees.

Find: The score $RF(P)$ and a full binary tree $S \in \mathcal{B}(X)$ such that $RF(P) = RF(P,S)$.

The cluster similarity problem

Here we first introduce the *cluster-similarity* measure, which is a measure complementary to the RF distance. We then introduce the *cluster-similarity* problem (Problem 4.2.5). This *cluster-similarity* measure has been implicitly used in [5], and Problem 4.2.2 is equivalent to the Problem 4.2.5.

Definition 4.2.3. [c-similarity]

Let S be an X-tree, and T be a Y-tree where $Y \subseteq X$. The *c(luster)-similarity from T to S* is defined as $R(T,S) := \left|\mathcal{C}(T) \cap \mathcal{C}\left(S_{|\mathsf{Le}(T)}\right)\right|$. Given a set P of partial X-trees, we define the *c-similarity from P to S* as $R(P,S) := \Sigma_{T \in P} R(T,S)$, and the *c-similarity of P* in the scope of $\mathcal{B}(X)$ as $R(P) := \max_{S \in \mathcal{B}(X)} R(P,S)$.

Corollary 4.2.4. *[c-similarity to RF conversion]*

For X-trees T and S, the conversion between the RF and c-similarity measures is $RF(T,S) = |\mathcal{C}(T)| + |\mathcal{C}(S)| - 2R(T,S)$.

Problem 4.2.5. [c-similarity]

Instance: A set P of partial X-trees.

Find: The similarity score $R(P)$, and a full binary tree $S \in \mathcal{B}(X)$ such that $R(P) = R(P,S)$.

Refinements for the cluster similarity problem

We are introducing two new problems that are refining the candidate trees for a solution of the c-similarity problem. First, we introduce the Problem 4.2.7, which is considering only candidate trees that have a cluster-presentation which is a subset of a given refining cluster set. Problem 4.2.9 is further narrowing down the refinement to candidate trees that can only have sibling clusters of a given set of sibling clusters. To guarantee solvability of these refined problems, we require that at least one candidate tree can be represented by the given refining sets, and call such refining sets *complete*. For each refined problem, we first introduce definitions necessary to state the problem, and then define the problem.

Definition 4.2.6. [cluster refined candidate trees]

Let X be a label set, and K be a set of subsets of X. We define $\mathcal{BC}(X,K) := \{T \in \mathcal{B}(X) \mid \mathcal{C}(T) \subseteq K\}$, and call K *cluster-complete* when $\mathcal{BC}(X,K) \neq \emptyset$. Given a set P of partial X-trees, we define the *c-similarity of P refined by K* as $RC(P,K) := \max_{S \in \mathcal{BC}(X,K)} R(P,S)$.

Problem 4.2.7. [cluster refined c-similarity]

Instance: Set P of partial X-trees and a cluster-complete set K of subsets of X.

Find: The similarity $RC(P,K)$ and a full binary tree $S \in \mathcal{BC}(X,K)$ where $RC(P,K) = R(P,S)$.

Definition 4.2.8. [sibling-refined candidate trees]

The *sibling presentation* of a full binary tree T is defined as $\mathsf{Sb}(T) := \{(A,B) \mid \exists \text{ siblings } u,v \in V(T) : \mathsf{Pa}_T(u) = \mathsf{Pa}_T(v), A := \mathcal{C}_T(u), B := \mathcal{C}_T(v)\}$. Let X be a label set, and L be a set of bi-partitions of subsets of X. We define $\mathcal{BS}(X,L) := \{T \in \mathcal{B}(X) \mid \mathsf{Sb}(T) \subseteq L\}$, and call L *sibling-complete* when $\mathcal{BS}(X,L) \neq \emptyset$. Given a set P of partial X-trees we define the *c-similarity of P refined by L* as $RS(P,L) := \max_{T \in \mathcal{BS}(X,L)} R(P,T)$.

Problem 4.2.9. [sibling refined c-similarity]

Instance: A set P of partial X-trees and a sibling-complete set L of bi-partitions of subsets of X.

Find: The similarity $RS(P,L)$ and a full binary tree $S \in \mathcal{BS}(X,L)$ such that $RS(P,L) = R(P,S)$.

4.3 Structural Properties of the C-Similarity Problems

The c-similarity problem and the refined c-similarity problems share an optimal substructure. Similar substructures have been shown for the gene tree parsimony problems [71, 79, 40]. Here, we first phrase this optimal substructure for the c-similarity measure, and next we follow up with the recurrences for each of the c-similarity problems.

Optimal substructure of the c-similarity problem

Definition 4.3.1. [restricted c-similarity]

Let S be an X-tree, T be a Y-tree where $Y \subseteq X$, and $Z \subseteq X$. The *c-similarity from T to S restricted by Z* is defined as $R_{|Z}(T,S) := \left|\mathcal{C}(T) \cap \mathcal{C}\left(S_{|Y \cap Z}\right)\right|$. Given a set P of partial X-trees we define (i) the *c-similarity from P to S restricted by Z* as $R_{|Z}(P,S) := \Sigma_{T \in P} R_{|Z}(T,S)$, and (ii) the *c-similarity of P restricted by Z* in the scope of $\mathcal{B}(X)$ as $R_{|Z}(P) := \max_{S \in \mathcal{B}(X)} R_{|Z}(P,S)$.

For any X-tree S with subtree $S_{|Z}$ the restricted c-similarity establishes the upper bound $RF(T,S) \leq |\mathcal{C}(T)| + |\mathcal{C}(S)| - 2R_{|Z}(T,S)$ for the Robinson-Foulds distance in a straightforward manner. Figure 4.3.1 shows the restricted c-similarity measure by example.

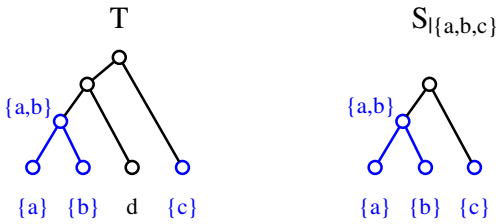


Figure 4.3.1 This example shows the c-similarity for the Y-tree T and X-tree S restricted to the set $Z := \{a,b,c\}$. The blue nodes indicate 4 common clusters $\{a\},\{b\},\{c\}$, and $\{a,b\}$, hence the restricted c-similarity is $R_{|Z}(T,S) = 4$. These clusters imply an upper bound on the Robinson-Foulds distance for every X-tree S containing the subtree $S_{|Z}$.

Proposition 4.3.2. *[optimal substructure]*

Let P be a set of partial X-trees, and S be a full binary X-tree such that $R_{|X}(P) = R_{|X}(P,S)$. Then $R_{|Z}(P) = R_{|Z}(P,S_v)$ where $Z = \mathsf{Le}(S_v)$ for any vertex $v \in V(S)$.

Proof. Let $V := S_v$. Assume for the purpose of a contradiction that $R_{|Z}(P) > R_{|Z}(P,V)$. Then, there must be a Z-tree U where $R_{|Z}(P,U) > R_{|Z}(P,S_v)$. Now, we construct the X-tree S' from S by replacing the Z-tree V with the Z-tree U. Let $W_{|X} := \mathcal{C}\left(S_{|X}\right) \setminus \mathcal{C}\left(V_{|X}\right)$, then we have $\mathcal{C}(S) = W_{|X} \dot{\cup} \mathcal{C}\left(V_{|X}\right)$ and $\mathcal{C}(S') = W_{|X} \dot{\cup} \mathcal{C}\left(U_{|X}\right)$, and this yields $R_{|X}(P,S) < R_{|X}(P,S')$

as follows.

$$
\begin{aligned}
& R_{|X}\left(P, S'\right) - R_{|X}\left(P, S\right) \\
&= \Sigma_{T \in P}\left(\left|\mathcal{C}\left(T\right) \cap \mathcal{C}\left(S'_{|\mathsf{Le}(T)}\right)\right| - \left|\mathcal{C}\left(T\right) \cap \mathcal{C}\left(S_{|\mathsf{Le}(T)}\right)\right|\right) \\
&= \Sigma_{T \in P}\left(\left|\mathcal{C}\left(T\right) \cap \left(W_{|\mathsf{Le}(T)} \dot{\cup} \mathcal{C}\left(U_{|\mathsf{Le}(T)}\right)\right)\right| - \left|\mathcal{C}\left(T\right) \cap \left(W_{|\mathsf{Le}(T)} \dot{\cup} \mathcal{C}\left(V_{|\mathsf{Le}(T)}\right)\right)\right|\right) \\
&= \Sigma_{T \in P}(\left|\mathcal{C}\left(T\right) \cap W_{|\mathsf{Le}(T)}\right| + \left|\mathcal{C}\left(T\right) \cap \mathcal{C}\left(U_{|\mathsf{Le}(T)}\right)\right| \\
&\qquad - \left|\mathcal{C}\left(T\right) \cap W_{|\mathsf{Le}(T)}\right| - \left|\mathcal{C}\left(T\right) \cap \mathcal{C}\left(V_{|\mathsf{Le}(T)}\right)\right|) \\
&= \Sigma_{T \in P}\left(\left|\mathcal{C}\left(T\right) \cap \mathcal{C}\left(U_{|\mathsf{Le}(T)}\right)\right| - \left|\mathcal{C}\left(T\right) \cap \mathcal{C}\left(V_{|\mathsf{Le}(T)}\right)\right|\right) \\
&= R_{|Z}\left(P, U\right) - R_{|Z}\left(P, V\right) \qquad \text{by } R_{|Z}\left(P, U\right) > R_{|Z}\left(P, V\right) \\
&> 0
\end{aligned}
$$

From $R_{|X}\left(P, S\right) < R_{|X}\left(P, S'\right)$ follows the contradiction $R_{|Z}\left(P\right) > R_{|Z}\left(P, V\right)$ as desired. □

Recurrences for the c-similarity problems

We describe the recurrences for the c-similarity problems introduced in Section 4.2. The recurrences in this section follow from the optimal substructure given in Proposition 4.3.2. Throughout this section let P be a set of partial X-trees, $Y \subseteq X$, and $Y \neq \emptyset$.

Definition 4.3.3. [cluster induced c-similarity score]

Let T be a Z-tree. We define $\Gamma\left(T, Y\right) := 1$, if $Y \cap Z \in \mathcal{C}\left(T\right)$, and $\Gamma\left(T, Y\right) := 0$, otherwise. Let P be a set of partial X-trees. We define $\Gamma\left(P, Y\right) := \Sigma_{T \in P}\Gamma\left(T, Y\right)$.

Proposition 4.3.4. *[c-similarity recurrence]*

$$
R_{|Y}\left(P\right) := \begin{cases} \Gamma\left(P, Y\right), & \text{if } \left|Y\right| = 1, \\ \Gamma\left(P, Y\right) + \max_{(A,B) \in \Pi(Y)}\left(R_{|A}\left(P\right) + R_{|B}\left(P\right)\right), & \text{otherwise,} \end{cases}
$$

where $\Pi\left(Y\right)$ *is the set of all non-trivial bi-partitions for* Y.

Definition 4.3.5. [restricted cluster refined c-similarity]

Let K be a cluster-complete set of subsets of X, and $Z \subseteq X$. The *c-similarity of* P *refined by* K*, and restricted by* Z is defined as $RC_{|Z}\left(P, K\right) := \max_{S \in \mathcal{BC}(X,K)} R_{|Z}\left(P, S\right)$.

Proposition 4.3.6. *[cluster refined recurrence]*

Let K be a cluster-complete set of subsets of X. Then, we have the following recurrence:

$$RC_{|Y}(P,K) := \begin{cases} \Gamma(P,Y), & \text{if } |Y| = 1, \\ \Gamma(P,Y) + \omega, & \text{otherwise,} \end{cases}$$

where $\omega := \max_{(A,B)\in\Pi(Y):\, \{A,B\}\subseteq K} \left(RC_{|A}(P,K) + RC_{|B}(P,K)\right)$.

Definition 4.3.7. [restricted sibling refined c-similarity]

Let L be a sibling-complete set of bi-partitions of subsets of X, and $Z \subseteq X$. The *c-similarity of P refined by L, and restricted by Z* is defined as $RS_{|Z}(P,L) := \max_{S\in\mathcal{BS}(X,L)} R_{|Z}(P,S)$.

Proposition 4.3.8. *[sibling refined recurrence]*

Let L be a sibling-complete set of bi-partitions of subsets of X. Then, we have the following recurrence:

$$RS_{|Y}(P,L) := \begin{cases} \Gamma(P,Y), & \text{if } |Y| = 1, \\ \Gamma(P,Y) + \omega, & \text{otherwise,} \end{cases}$$

where $\omega := \max_{(A,B)\in L: A\dot\cup B=Y} \left(RS_{|A}(P,L) + RS_{|B}(P,L)\right)$.

4.4 Dynamic Programming for RF and C-Similarity Supertree Problems

Following from Corollary 4.2.4, a solution for the c-similarity supertree problems also provides a solution for the equivalent RF supertree problems. Here we present a dynamic programming (DP) solution for the c-similarity problem and the refined c-similarity problems introduced in Section 4.2. First, we describe the algorithm for the recurrence of the sibling refined c-similarity problem, and then based on this algorithm, we follow up the solutions for the other c-similarity problems.

Correctness: We show that Algorithm 4.1 computes the recurrence in Proposition 4.3.8 for Problem 4.2.9.

Proof. Let S be a solution X-tree for Problem 4.2.9, i.e. a full binary tree $S \in \mathcal{BS}(X,L)$ such that $RS(P,L) = R(P,S)$. Then $\mathcal{C}(S) \subseteq L$, so $RS_{|Y}(P,L)$ is defined for $Y \in \mathcal{C}(S)$.

We show that $RS_{|Y}(P,L) = \mathcal{R}[Y]$ for all $Y \in \mathcal{C}(S)$. For $|Y| = 1$, which are trivial clusters, $\mathcal{R}[Y] \leftarrow RS_{|Y}(P,L)$ is assigned in Part 1. Let Y be a cluster with the smallest number of

Input:
Set P of partial X-trees
Sibling-complete set L of bi-partitions of subsets of X

Output:
The table $\mathcal{R}$ of restricted c-similarity scores $R_{|Y}(P, L)$,
where Y is an element of the X-complete set $\{A \cup B | (A, B) \in L\}$

Algorithm:
Part 1. Initialize the DP table $\mathcal{R}$ for trivial clusters.
$\mathcal{R} \leftarrow \emptyset$ with the default value of $-\infty$ for unassigned rows
for $v \in X$ **do**
    $\mathcal{R}[\{v\}] \leftarrow \Gamma(P, Y)$
end for

Part 2. Compute the DP table $\mathcal{R}$ for non-trivial clusters.
$(A_1, B_1), (A_2, B_2), \ldots, (A_t, B_t)$ is a sorted list of L
    such that $|A_i \dot{\cup} B_i| \leq |A_{i+1} \dot{\cup} B_{i+1}|$ for $1 \leq i < t$ // (1)
for $i \leftarrow 1$ **to** t **do**
    $\mathcal{R}[A_i \dot{\cup} B_i] \leftarrow \max(\mathcal{R}[A_i \dot{\cup} B_i],\ \mathcal{R}[A_i] + \mathcal{R}[B_i] + \Gamma(P,\ A_i \dot{\cup} B_i))$ // (2)
end for
return $\mathcal{R}$

Algorithm 4.1 This dynamic program (DP) solves the sibling refined c-similarity problem, Problem 4.2.9 by computing the DP table $\mathcal{R}$.

taxa such that $|Y| > 1$ and $\mathcal{R}[Y] \neq RS_{|Y}(P, L)$. Then, $\exists (A, B) \in L$ such that $A \dot{\cup} B = Y$ and $RS_{|Y}(P, L) = RS_{|A}(P, L) + RS_{|B}(P, L) + \Gamma(P, Y)$, otherwise $|Y| = 1$ or $Y \notin \mathcal{C}(S)$. $A \neq \emptyset$ and $B \neq \emptyset$ by the definition of sibling clusters. Then, since $\mathcal{R}[Y] \neq RS_{|Y}(P, L)$, either $\mathcal{R}[A] \neq RS_{|A}(P, L)$ or $\mathcal{R}[B] \neq RS_{|B}(P, L)$, but $|A| < |Y|$ and $|B| < |Y|$, which contradicts the assumption that Y is the smallest one. Next, let us assume the recursion is not computed in order; that means for some $A \dot{\cup} B \in \mathcal{C}(S)$ the assignment $\mathcal{R}[A \dot{\cup} B] \leftarrow \mathcal{R}[A] + \mathcal{R}[B] + \Gamma(P,\ A \dot{\cup} B)$ occurred before $\mathcal{R}[A] \leftarrow R_{|A}(P, L)$ or $\mathcal{R}[B] \leftarrow R_{|B}(P, L)$. Without loss of generality let it be $\mathcal{R}[A]$. However, $|A| < |A \dot{\cup} B|$ and the traversal in Part 2 strictly enforces that sequence in Line (1), so the assignment $\mathcal{R}[A] \leftarrow RS_{|A}(P, L)$ occurs before the assignment $\mathcal{R}[A \dot{\cup} B] \leftarrow \mathcal{R}[A] + \mathcal{R}[B] + \Gamma(P,\ A \dot{\cup} B)$. Therefore, $RS_{|Y}(P, L) = \mathcal{R}[Y]$ for all $A \dot{\cup} B$ of the recursion, and Algorithm 4.1 computes the recurrence correctly for $R_{|Y}(P, L)$ for all $Y \in \mathcal{C}(S)$. □

Theorem 4.4.1. *[sibling refined c-similarity]*

Let P be a set of partial X-trees and L be a sibling-complete set of bi-partitions of subsets

of X. The c-similarity problem, Problem 4.2.9, can be solved in $O(tnm)$ time, where $t := |L|$, $n := |X|$, and $m := |V(P)|$.

Proof. Let S be the sibling refined RF supertree that is constructed by following the recursive dynamic programming structure of the refined c-similarity recurrence. Then, S is the solution for Problem 4.2.9.

Complexity: The computational complexity for Algorithm 4.1 is dominated by computing the restricted c-similarity for non-trivial clusters. Part 1: Computing the restricted c-similarity for trivial clusters takes $O(m)$ time, by counting the different leaf nodes in P. Part 2: Sorting Q takes $O(tn)$ time. The access time for an element in $\mathcal{R}$ is $O(n)$ and $\Gamma(P, A \dot{\cup} B)$ is computed in $O(nm)$ time, so computing the restricted c-similarity for a non-trivial cluster in Line 2 takes $O(nm)$ time in a loop of $O(t)$ iterations. Therefore, the computational complexity bound of Algorithm 4.1 is $O(tnm)$ time. □

The dynamic programming algorithm for Theorem 4.4.1 can be adapted to also solve Problem 4.2.7 and Problem 4.2.5. Next, we construct different sibling-complete sets for each problem. Table 4.1 summarizes the different inputs.

Theorem 4.4.2. *[cluster refined c-similarity]*

Let P be a set of partial X-trees and K be a cluster-complete set of bi-partitions of subsets of X. The cluster refined c-similarity problem can be solved in $O\left(c^2 nm\right)$ time, where $c := |K|$, $n := |X|$, and $m := |V(P)|$.

Table 4.1 Inputs for RF supertree problems

supertree Problem	sibling-complete set	size
(a) exact	$\{\Pi(Y) \mid Y \in \mathcal{P}(X),\ Y \neq \emptyset\}$	$O\left(3^{\lvert X\rvert}\right)$
(b) cluster refined	$\{(A, B) \mid A, B \in K,\ A \cap B = \emptyset\}$	$O\left(\lvert K\rvert^2\right)$
(c) sibling refined	L	$\lvert L\rvert$

The input for Algorithm 4.1 for the exact and refined c-similarity supertree problems, (a) Problem 4.2.5, (b) Problem 4.2.7, and (c) Problem 4.2.9. X is the taxa set, $\Pi(Y)$ the set of all non-trivial bi-partitions for Y, $\mathcal{P}(X)$ be the power set of set X, K is a cluster-complete set of subsets of X, and L is a sibling-complete set of bi-partitions of subsets of X.

Proof. Let S be a solution X-tree for Problem 4.2.7, i.e. a full binary tree $S \in \mathcal{BC}(X, K)$ such that $RC(P, K) = R(P, S)$. It follows that $\mathcal{C}(S) \subseteq K$, so for each pair of siblings $u, v \in V(S)$ there exists (A, B) such that $A, B \in K$ and $\mathcal{C}_S(u) = A$ and $\mathcal{C}_S(v) = B$. Construct $L := \{(A, B) | A, B \in K,\ A \cap B = \emptyset\}$ for the input of Algorithm 4.1, and it follows that $\mathsf{Sb}(S) \subseteq L$.

Complexity: The size of L is $O(c^2)$. It follows, the computational complexity for Problem 4.2.7 is bounded by $O(c^2 nm)$ time. □

Theorem 4.4.3. *[c-similarity]*

Let P be a set of partial X-trees. The c-similarity problem can be solved in $O(3^n nm)$ time, where $n := |X|$ and $m := |V(P)|$.

Proof. Let S be a solution X-tree for Problem 4.2.5; i.e. a full binary tree $S \in \mathcal{B}(X)$ such that $R(P) = R(P, S)$. Let $\mathcal{P}(X)$ be the power set of set X. Construct $L := \{\Pi(Y) | Y \in \mathcal{P}(X), Y \neq \emptyset\}$ for the input of Algorithm 4.1. For all $T \in \mathcal{B}(X)$, $\mathsf{Sb}(T) \subseteq L$. It follows, that $\mathsf{Sb}(S) \subseteq L$.

Complexity: There are $\frac{1}{2}(3^n - 2^{n+1} + 1) = O(3^n)$ unique sibling clusters in L. It follows, the computational complexity for Problem 4.2.5 is bounded by $O(3^n nm)$ time.

Currently, the best-known (naive) exhaustive search algorithm for the RF supertree problem requires searching through all possible $(2n-3)!!$ rooted supertrees. Thus, the best naive solution for the RF supertree problem is solvable in $O((2n-3)!! nm)$ time. We obtain the speed up of $\Theta\left(\sqrt{n^n}/3^n\right)$ as follows:

$$\begin{aligned} \frac{(2n-3)!! nm}{3^n nm} &= \frac{(2n-3)!!}{3^n} \\ &\leq \frac{n^{n/2} - 4}{3^n} \qquad \backslash\backslash \text{ for } n > 0 \\ &= \Theta\left(\sqrt{n^n}/3^n\right) \end{aligned}$$

□

Different supertree objectives

Our DP algorithms for the efficient RF supertree search can also be directly applied to other supertree objectives, these include gene duplications and loss, deep coalescence, and Maximum

Parsimony (MP) for the MRP problem. The requirement on the supertree objective is the objective specific definition of the restricted score, equivalent to the restricted c-similarity for RF. That implies: given a subtree $S_{|Z}$, the Z-restricted score has to invoke an upper bound on the score if the subtree $S_{|Z}$ is contained in the supertree. For gene duplication and loss, this restricted score is the number of duplications and losses induced by $S_{|Z}$, for deep coalescence the restricted score is the number of embedded lineages induced by $S_{|Z}$, and for MP the restricted score is the number of character state changes induced by $S_{|Z}$. The specific definitions of the restricted scores for these objectives are straightforward, and omitted for brevity.

4.5 Experiments

We first examined if knowledge-enhanced searches can improve upon solutions from RF supertree heuristics using the SPR local search [5]. We used three large data sets, one published data set from marsupials [20] and two unpublished data sets from the gymnosperm and Saxifragales plant clades. The gymnosperm trees were made from RAxML analyses of the gene alignments found in [19], and the Saxifragales trees were made with RAxML analyses of gene alignments assembled from GenBank. The RAxML analyses all used the GTRCAT model, and the tree data sets will be available from the dryad data repository (`http://datadryad.org`). In all the experiments, we first ran the SPR-heuristic to obtain a set of RF supertrees. For each data set, we used 10 different starting trees, built by random stepwise addition, and for each starting tree, we repeated the RF-SPR using a ratchet heuristic (see [5]) 25 times for each search. This resulted in 250 RF supertrees for each data set. These analyses took between 6.5 hours (for the marsupial data set) and 10 days (for the gymnosperm data set).

We ran knowledge-enhanced searches using the 250 RF supertrees from the SPR analysis as the constraint set. The cluster refinement method finished only for the marsupial data set (in 5 days). However, the sibling refinement method runs took between 18 seconds (for the marsupial data set) and 1.5 minutes (for the gymnosperm data set). In all three data sets, the constrained RF search improved the RF score of the best RF supertrees from the SPR search. The improvements were between 1.06 and 5.32% (see Table 4.2). This experiment demonstrates that the solutions to the RF supertree problem from the SPR local search are

suboptimal, and given the proper set of knowledge-enhanced supertrees, the sibling refinement method can rapidly improve upon RF supertrees made with SPR heuristics.

Table 4.2 The improvement of the sibling refinement method on RF supertrees constructed by the SPR local search heuristic.

Data Set	RF Method	rooted RF Distance	Improvement
Saxifragales	SPR local search	2308	48
(959 taxa; 51 trees)	Refinement method	2260	2.08 %
Marsupial	SPR local search	1510	16
(272 taxa; 158 trees)	Refinement method	1492	1.06 %
Gymnosperm	SPR local search	4343	231
(950 taxa; 78 trees)	Refinement method	4112	5.32 %

Next, we examined the performance of our exact RF supertree solution on a data set with 699 gene trees representing 12 plant taxa. The taxa include an outgroup moss, *Physcomitrella patens*, nine gymnosperm taxa including cycad Cycas rumphii, Gnetales taxa *Gnetum gnemon* and *Welwitschia mirabilis*, and the conifers *Cryptomeria japonica*, *Pseudotsuga menziesii*, *Picea glauca*, *Picea sitchensis*, *Pinus pinaster*, and *Pinus taeda*, and finally two angiosperm (*Arabidopsis thaliana* and *Oryza sativa*). To build the gene trees, we obtained amino acid alignments for genes from Phytome v.2, an online comparative genomics database for plants [42]. We sampled 699 genes that had exactly one sequence from at least four of our selected taxa. We built maximum likelihood gene trees from each alignment RAxML-VI-HPC v.2.2.3 [65] with the JTTCAT amino acid substitution model [44]. The gene trees will be available on the dryad data repository. It took 13.2 minutes to compute the exact RF solution for this data set on a Macintosh Powerbook with a 2 GHz Intel® Core™ 2 Duo processor and 2 GB memory. The RF score was 528, and the supertree, which indicated the Gnepine phylogenetic hypothesis for seed plants, was consistent with many recent phylogenetic studies. While this represents the largest empirical data set with an exact solution for the RF supertree problem, we also were able to compute exact solutions for data sets with up to 17 taxa and 100 random topologies in 9 hours.

4.6 Conclusion

Knowledge-enhanced RF supertree problems can rapidly construct supertrees from sets of existing, possibly contradictory, phylogenetic relationships. This approach can be used to obtain an exact solution to a supertree problem, while ensuring that the solution only includes the given phylogenetic relationships.

Our experiments demonstrated that, not only do our exact algorithms far exceed the best-known (naïve) solution for solving the RF supertree problem, the knowledge-based supertree searches provide a fast and simple approach to improve upon suboptimal solutions for local search heuristics for the RF supertree problem. Although we have focused on algorithms for RF supertrees in this study, we demonstrated that our algorithms are also adaptable for other supertree problems like the gene duplication, duplication and loss, deep coalescence, and MRP problems.

Our results suggest several future directions for research. Although the knowledge-enhanced supertree analyses can construct extremely large supertrees in a reasonable amount of time, the quality of the resulting supertrees depends on the provided phylogenetic relationships. In our experiments, acquiring the set of phylogenetic relationships took far longer than the knowledge-enhanced RF supertree search. Thus, the key to the application of knowledge-enhanced searches is the ability to build rapidly many high quality relationships, and our future work will explore best methods to do this.

CHAPTER 5. Visual Exploration for Gene/Species Tree Reconciliation

André Wehe, J. Gordon Burleigh, and Oliver Eulenstein

Abstract

Gene sequences contain phylogenetic information. Unfortunately, this information is not always the same as the phylogeny of species from which the genes were sampled. Encoded in genes are not only speciation events, but in addition, genes have their own complex histories with a record of genomic events such as gene duplications, gene losses, and horizontal gene transfers. It is necessary to account for these events when explaining these evolutionary differences: the gene/species tree reconciliation. With more, newly available phylogenetic information, the reconciliations become often very complex, and analyzing these complex, large-scale reconciliations can be very difficult with standard visualization methods. We developed a visual, interactive method for exploring complex, large-scale reconciliations and the analysis of gene duplications, gene losses, and deep coalescence.

5.1 Introduction

The gene/species tree reconciliation explains incongruence among gene and species trees with genomic events such as gene duplications, gene losses, and horizontal gene transfer. There is continuous interest in gene/species tree reconciliation (e.g., [26, 24, 63, 54]], and the NSF funded project iPlant http://www.iplantcollaborative.org). With the rapidly growing availability of genomic phylogenetic data and the quickly maturing methods for large-scale inference of gene trees and supertrees, large-scale gene/species tree reconciliations become inevitable. This raises the need for methods that allow the analysis of large-scale reconciliations.

However, large-scale reconciliation is often very complex, and standard methods that allow a detailed analysis, such as the reconciliation tree and the embedded reconciliation tree, were originally designed for small reconciliations and are in practice only of limited use for large-scale reconciliations. We developed an interactive method that allows the exploration of complex, large-scale reconciliations on gene duplications, gene losses, and deep coalescence.

There are multiple challenges in gene/species tree reconciliations, and several tools for the visualization and analysis of reconciliations have been developed. For instance, iGTP [22] allows the construction and casual analysis of species trees on gene duplication, gene loss, and deep coalescence, NOTUNG [24] allows the analysis of unresolved gene or species trees on the duplication-loss model, PrIMETV [63] produces visually appealing embedded reconciled trees, and Mesquite [48] is a modular framework that includes modules for small-scale reconciliation. However, none of them has been designed for the detailed analysis of very complex reconciliations of very large species trees and multiple gene trees. Standard visualization techniques for gene/species tree reconciliations can grow quadraticly in size, because each gene duplication presents a new, possibly different, evolution of the gene. Thus, reconciliations that include many gene duplications can quickly become crowded with genomic events and parallel phylogenies, thus such an analysis becomes practically impossible with standard techniques. For instance, modern methods allow the reconstruction of species trees with thousands of taxa (e.g., [74, 4, 8]), but the analysis of gene duplication events within the recent history of a species might involve many genes in different clades of multiple large gene trees of thousands of genes.

Our contribution: Our interactive method displays standard cladograms of the species and gene trees, and we then project an overview of the reconciliation on the trees. In these overviews we allow for an interactive, more detailed exploration and analysis of the reconciliations. For this, elements of interest can be selected, such as species or genes, and then events that involve the selected elements can be explored across all input trees. This design is scalable, thus our method allows the analysis and exploration of complex, large-scale reconciliations of a species tree and multiple gene trees. We apply our interactive method to gene tree parsimony, where we demonstrate the use of our method for visual tree refinement. We then demonstrate

the exploration and analysis of a larger vertebrates reconciliation.

5.2 Reconciliation

The reconciliation is a map between a gene tree and a species tree with gene duplications and losses being placed to explain any incongruence between the trees. Figure 3.2.1 in Chapter 3 shows an example of a small reconciliation on two classic visualization techniques: The reconciled tree and the embedded reconciled tree. The reconciled tree is a tree of genomic events, and it expresses deviating evolutions in separate clades. The embedded reconciled tree embeds the genomic events and gene tree edges into edges of a scaffold species tree. For instance, the software GeneTree [54] uses reconciled trees for its analysis, and the program PrIMETV [63] is designed to produce an embedded reconciled tree.

5.3 Design

In this section we formulate the challenges of reconciliations and describe our design of the interactive visualization. Next, we describe challenges of gene tree parsimony and depict how to apply our visual interactive method to a tree refinement technique.

5.3.1 Visualization Design

The ideal reconciliation visualization would display very large and complex reconciliations in great detail, and the analysis would be easy. Of course, such an ideal visualization is imaginary. Large reconciliations are very complex, and this makes it impractical for displaying with standard visualization methods. In particular, gene duplications expand the size of reconciliations by introducing deviating evolutions for genes. Every gene duplication effectively introduces a new phylogenetic subtree that presents a deviating evolution for the duplicated gene. With many gene duplications, the complexity of reconciliations can grow quadraticly with the taxa size. Even for slightly larger trees this can quickly result in very big and complex reconciliations. Many modern data sets are of larger scale, and their reconciliation is complex. However, many existing tools are relying on standard reconciliation visualization techniques

and are designed for smaller reconciliations only. One approach to overcome this limitation is using a cumulative overview that expresses genomic events as quantities throughout the trees (e.g., [22, 54]). For instance, the total number of gene duplications that occurred in the immediate history of species is included. However, this hides necessary details of events, such as the exact genes that duplicated in a species. A better design would have both an overview display and a detailed view of genomic events. While our design follows the overview display of events, we extended the design with an interactive visualization that separates events and shows necessary details. While it might be useful to analyze all events simultaneously or single events in detail, scientists are also interested in analyzing particular sets of events collectively. Our design allows the selection of sets of species, genes, and edges that are of interest. When exploring different trees, this selection, their related genomic events, and their embeddings are tracked.

Event identification

To identify species and genes that have a history of gene duplication or gene loss, we deduce a higher-level, quantitative interpretation from the reconciliation. We refer to it by overview display. We indicate genes if they were duplicated or lost, and we indicate species where genes duplicated or were lost. This interpretation is then displayed on the gene and species tree. Nodes in the gene tree represent genes, and nodes in the species tree represent species. Leaves of the trees are species and genes that are currently in existence, whereas internal nodes are ancestral species or genes.

The reconciliation determines where and when these gene duplications and gene losses occurred. For gene duplication, we indicate the most recent species that could contain the gene duplication, and for gene loss we indicate the oldest species where there is no longer any evidence of the existence of that gene. The reconciliation is restricted to the sampled species only; that means, technically, the species tree is reduced to the gene trees taxon set prior to computing the reconciliation.

Calculating these events is formulated in terms of the mapping from the nodes in the gene tree to the LCA (least common ancestor) nodes in the species tree. This LCA mapping sets

the bound on how recent genes were still present in species before they evolved further. So, this also implies bounds on how recent events could have occurred. An event is then placed at the LCA node.

Gene duplication is formulated as follows: given a gene node g in the gene tree and its children g' and g'', let s, s', s'' be the LCA species nodes for g, g', g'', respectively. If $s = s'$ or $s = s''$, then g is duplicated and the most recent species that could contain the gene duplication is s.

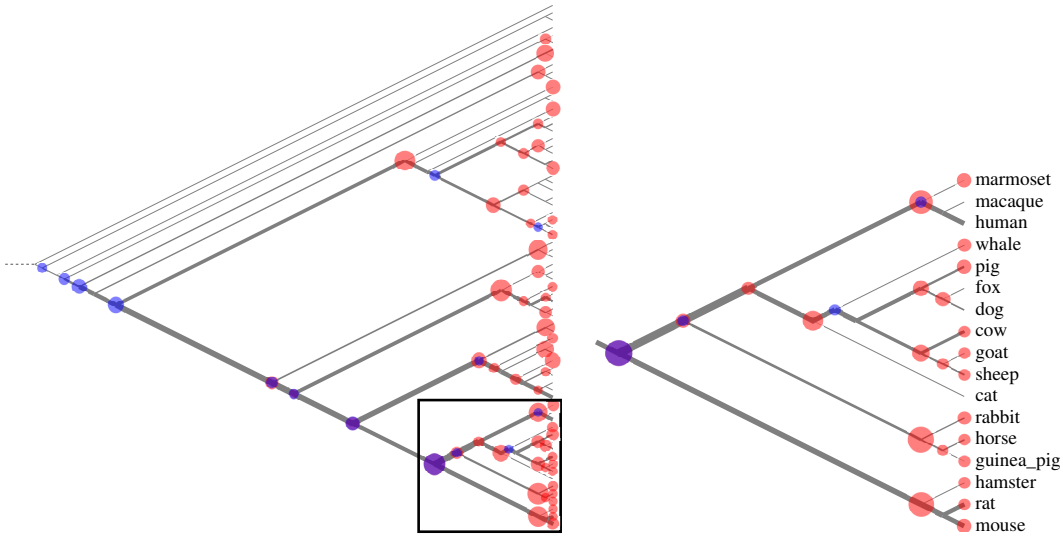
Gene loss is formulated as follows: given a gene node g in the gene tree and its child g', let s and s' be the LCA species nodes for g and g', respectively. Let T be a subtree forking of the path from s to s', then g is assumed to be lost in the root node of T.

The embedding of the gene trees into the species tree shows the number of parallel embeddings in edges. We indicate the number of parallel embeddings with the thickness of the edges of the species tree. The sum of all parallel-embedded edges is the deep coalescence score.

Visual metaphor

Our layout displays binary trees in a slanted cladogram with a horizontal timeline going from the past (left side) to the present (right side). Genes and species involved in events are colored; nodes are colored blue for gene duplications and red for gene loss. The thickness of the node shows the relative number of events. The thickness of an edge shows the relative number of parallel embeddings. Figure 5.3.1 depicts an example of the overview display of a species tree and one of its subtrees. Shifting and zooming allows for the display of very large trees.

The analysis of genomic events across multiple gene trees and the species tree is rather complicated. We assign each tree a sheet of virtual paper in a separate window, and allow displaying multiple trees concurrently. This allows for a very flexible layout where the selected trees can be freely arranged on the desktop, and all trees relevant for an analysis are displayed. Thsi layout enables the standard side-by-side analysis of the species tree and one gene tree, but it also allows the efficient analysis across the species tree and multiple gene trees.



(a) (b)

Figure 5.3.1 (a) Gene duplication and gene loss in a subtree of the vertebrate species tree. (b) The lower clade with species labels. Blue nodes are species where genes duplicated and red nodes are species where genes were lost. Thick edges contain parallel embeddings.

5.3.1.1 Interaction

The complexity of larger reconciliations not only hampers a functional, comprehensive, and detailed display of all genomic events, it also enormously complicates the analysis of events. Scientists are often interested in investigating particular genomic events or sets of events. Events that are not relevant for this analysis are mostly interfering.

Our design defaults to a display that provides a quantitative overview of genomic events and embeddings. The overview works for large trees, but hides details of the reconciliation. Simultaneously, our design reveals more details of the reconciliation on demand. This is by selecting species, genes, or edges in the trees. These details are the number of gene duplications, the number of gene losses, and the number of embeddings. Unlike the interactive limitations of visualizations on single trees, our design displays the details across multiple trees. For instance, selecting a duplicated gene in the gene tree will indicate the species in the species tree where the duplication event occurred, and vice versa. Table 5.1 and Table 5.2 show a complete list of the interactive display of the reconciliation across trees.

element pointed at in the gene tree	elements indicated in the species tree
a node g (gene)	the LCA of g
an edge e	the embedded path of e
a gene g that duplicates	the species s where g duplicates genes that map
a gene g that got lost	species where g got lost, the paths where g did not get lost

Table 5.1 Interactive display of the reconciliation from the gene tree onto species tree.

element pointed at in the species tree	elements indicated in the gene trees
a node s (species)	the genes where the LCA mapping is s, the embedded edges passing through s
an edge e	the embedded edges passing through e
a gene duplication event in species s	the genes that duplicate in s
a gene loss event in species s	the genes that got lost in species s

Table 5.2 Interactive display of the reconciliation from the species tree onto the gene trees.

5.3.2 Challenges with Gene Tree Parsimony

The gene tree parsimony presents a number of challenges. The reconciliation explains incongruence between the species and gene trees. However, the correct species and gene trees are often unknown. There are exponentially many different possible trees, and it is not possible to analyze all of them for many taxa, but trees can often be estimated. However, inaccuracy in the trees can largely disrupt a correct reconciliation by introducing false genomic events and misplacing events in the species tree. A detailed analysis of the reconciliation can reveal errors in the trees.

Another challenge is to determine the correct phylogeny of gene or species trees, when multiple competing phylogenies exist. Here, an overview might be insufficient, but a detailed analysis of genomic events and the embedding could decide the true hypothesis. However, analyses on complex reconciliations are particularly difficult. Complex reconciliations contain a large number of gene duplications, gene losses, and embeddings of multiple gene trees.

The complexity makes an analysis using classical visualizations cumbersome, and computational methods may hide necessary, basic details. It is often not possible to limit the analysis to a smaller set of genes and species, because many genomic events are evolutionarily not in-

dependent from each other. For instance, misplaced gene duplications can cause additional artificial gene duplications in ancestral species. Such complex reconciliations make an analysis difficult. While new phylogenetic data is available for more genes and species, the analysis on complex reconciliations has become an increasingly common challenge.

Tree refinement

One problem scientists are interested in is the effect that changes on the species and gene trees have on the reconciliation. The evolution of some groups of species or genes is often envisioned differently than current assumptions suggest. A good approach is to modify the current assumed trees; then the reconciliation can easily be computed and genomic events can be placed appropriately. However, for complex reconciliations this can become difficult. Therefore, it is important to create an intuitive, interactive visualization that allows a quick and effective exploration of species and gene trees, and analysis of the reconciliation. For a visual tree refinement design it is essential to be able to monitor the effects that changes on the trees have on the reconciliation. We allow the concurrent display of multiple gene trees and the species tree. Our design allows for the editing of species and gene trees (i) by pruning and regrafting of subtrees and (ii) by re-rooting of subtrees. The reconciliation and counts of gene duplications, gene losses, and parallel embeddings (the deep coalescence score) are immediately updated after each change in the trees. During tree refinement the selection of genes, species, and branches remains unchanged. Only the related events and embeddings are updated. This allows the monitoring of selected events while exploring different trees. The interface applies a simple drag-n-drop technique to describe tree edit operations, that is, dragging the root node r of a subtree T_r into an edge (u, v) of the tree T. We differentiate the type of edit operation with the destination edge (u, v) in relation to the subtree T_r: if (u, v) is not contained in T_r, then T_r will be pruned and regrafted into the edge (u, v), otherwise if (u, v) is contained in T_r, then T_r will be re-rooted with $\{u, v\}$ as children of the new root node. This provides a simple, quick, and intuitive interface for both types of tree edit operations.

5.4 Implementation

For our implementation, gene and species trees are rooted, fully binary trees. We display trees in a sideways oriented cladogram and apply 2D hardware acceleration of OpenGL for fast and smooth navigation within trees. Trees are displayed on virtual sheets of paper, and the mouse allows the easy navigation on these sheets (sliding of the paper, scaling of the tree, and selecting of nodes and edges).

Computing the reconciliation has been historically slow, but algorithmic improvements (e.g., [12]) allow computing it in linear time. In particular, tree exploration requires frequent new reconciliations. We implemented these advanced algorithms to enable a fast tree exploration even for large reconciliations.

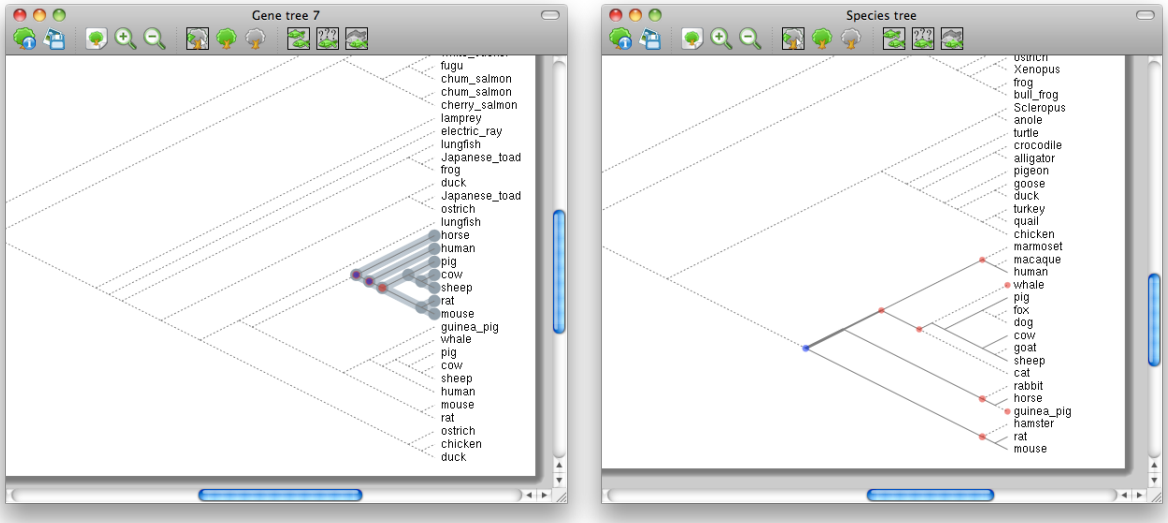
Since scientists work with all standard operating systems, we found it to be considerate to make our tool available for MS Windows, Mac OS X, and Linux. This allows a seamless workflow with third party software. The software is available by request from the authors.

5.5 Example: Vertebrate Data Set

In this section we show by example how to explore reconciliation in trees, and we show how our visualization can be used to analyze genomic events. The example data set is a species tree of 71 vertebrate taxa and a set of 8 gene trees. This data set is sufficiently large that a standard visualization technique becomes challenging. Figure 5.3.1 depicts an overview of the genomic events in the species tree.

5.5.1 Exploration of the Reconciliation

Figure 5.5.1 shows an example where the genomic events caused by a particular subtree of genes are being explored. Any set of nodes and edges can be selected and tracked. In this example we select all nodes and edges of a subtree T. Displayed are only genomic events manifested by the evolution of genes in T. For the tree refinement method, any change to the trees will update these events.



(a) (b)

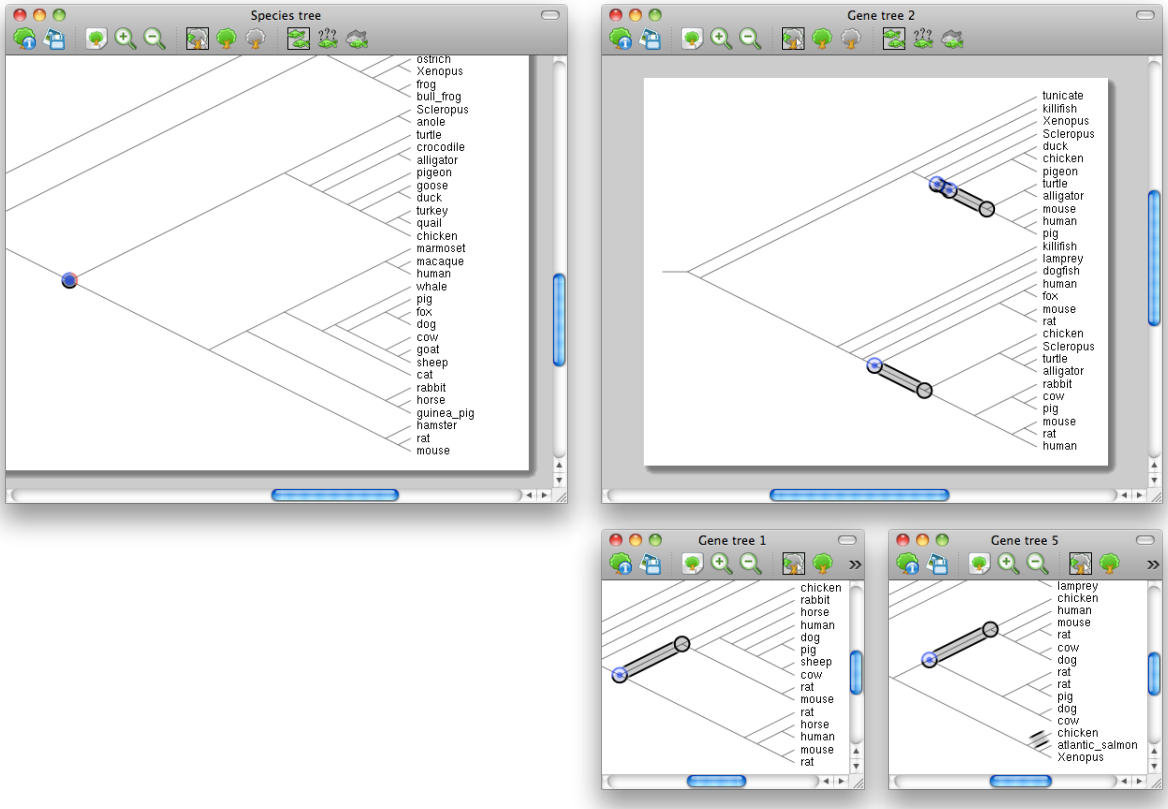
Figure 5.5.1 This is an example of tree exploration where the reconciliation of a selected subtree is being explored as it would be displayed in our application. (a) A subtree T in a gene tree is selected. (b) The species tree is displayed with the embedding, gene duplications, and gene losses only for T.

5.5.2 Analysis of Genomic Events and Embeddings

Basically there are two types of interactive analysis: one focuses on elements in the species tree, the other focuses on elements in a gene tree. For instance, by selecting a species, all relevant genomic events and embeddings are displayed (see Table 5.1 and Table 5.2 for details). Next, we describe examples that demonstrate the analysis concept for gene duplication, gene loss, and deep coalescence.

5.5.2.1 Gene duplication

Figure 5.5.2 depicts an example that focuses on a species s and then identifies all the genes that duplicate in the recent history of s. Gene tree 1, 2, and 5 contain genes that duplicate in s. These genes are colored blue in these trees. Gene tree 1 and 5 each contain one gene that duplicates in s, and gene tree 2 contains 3 genes that duplicate in s. The display also reveals in black the genes and edges that are embedded in species s. This embedding reveals that there was one gene, the upper one in gene tree 2, that duplicated twice in close iteration, whereas the others were single duplications.



(a) (b)

Figure 5.5.2 This is an example of the interactive display for a species tree and multiple gene trees, as it would be displayed in our application. (a) Selected is a species s with five gene duplications. (b) Three gene (sub)-trees with highlighted genes and edges that explain the reconciliation and the gene duplications for species s: the LCA of black nodes is s, black edges are embedded in the species tree with s along their paths, and blue nodes are genes that duplicate in the species s.

5.5.2.2 Gene loss

Figure 5.5.3 describes an example that focuses on a gene g and then identifies all species where g was lost. The red paths show the embedding of the edges $(g, human)$ and (g, fox) and indicate the species that had the gene g. The gene g may or may not have been lost in any of the species that fork off this red path. Since the gene tree does not contain any data on these animals, nothing can be said about whether or not they have the gene. The red circles indicate species where g could have been lost, and there is evidence that g got lost in one of its descendant species.

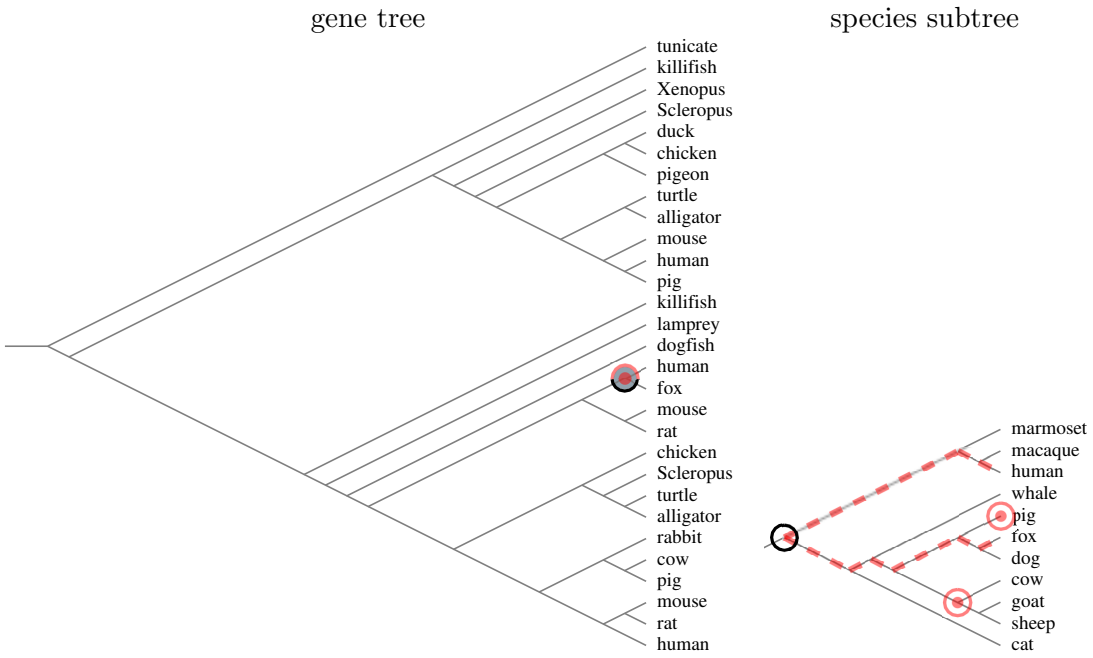


Figure 5.5.3 This is an example of gene losses for a gene. (a) A selected gene g in the gene tree with four losses. (b) A species subtree with highlighted species and paths that explain the reconciliation and the gene losses for gene g: the black circle is the LCA of g in the species tree, the red paths shows the species that had the gene g, and the red circles show species where g got lost.

5.5.2.3 Deep coalescence

The embedded reconciled tree embeds gene trees in a species tree. The deep coalescence score is computed by counting the number of parallel embeddings of gene tree edges in each edge of the species tree. Figure 5.5.4 depicts an example that focuses on an edge e in the species tree and reveals the gene tree edges that are embedded in e. Additionally, the edge $(g, human)$ was selected in the gene tree, and the dotted line in the species tree shows the embedding of $(g, human)$.

5.6 Conclusion

Visualization has been a central tool for analyzing and describing gene/species tree reconciliations. While standard visualization techniques have been the de facto method for small-scale reconciliations, they have been only limited useful on large-scale reconciliations. In this paper

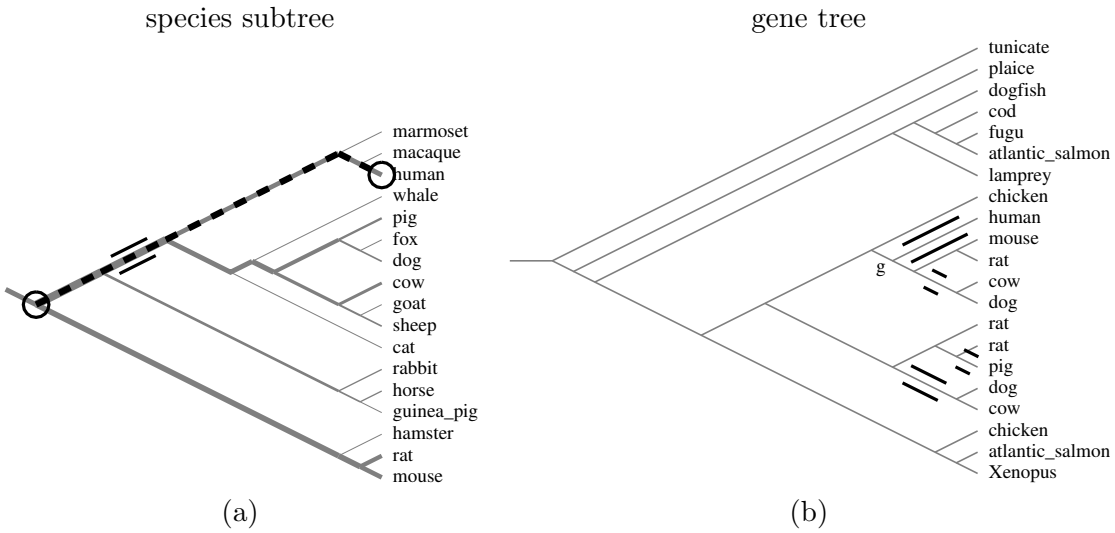


Figure 5.5.4 This is an example of parallel embeddings, e.g., for analyzing deep coalescence. (a) A selected edge e in the species tree with multiple parallel embeddings, and the embedding of $(g, human)$. (b) A gene tree and the indicated edges whose embeddings pass through e.

we introduce a novel interactive method that allows in-depth, visual analysis of large-scale reconciliations. Our visualization depicts a high-level view of genomic events and embeddings and then adds an interactive component that allows the detailed analysis of the reconciliation. Our method enables the exploration of complex, large-scale reconciliations and the selective analysis of the reconciliation down to genomic event level. We demonstrate the applicability of our method on visual tree refinement by tracking relevant events while refining trees. We demonstrate the exploration of the reconciliation and the analysis of gene duplication events, gene loss events, and deep coalescence by example on a larger vertebrate data set where previous standard visualizations would have reached their limits. With new technical advancements, our visual method could be adapted to even more intuitive peripheral input devices such as touch screens and tracking devices.

CHAPTER 6. General Conclusion

This thesis addresses three different problems for phylogenetic supertree construction: the high algorithmic time complexities of the gene duplication problem, the quality of supertrees, and the challenges of analyzing and exploring complex supertrees. Local searches have already proven to be invaluable for the gene duplication problem [18]. The high time complexities prohibited truely large-scale supertree reconstructions that are needed for reconstructing the tree of life. Our new algorithmic advancements in Chapter 2 and the parallelization in Chapter 3 enable supertree reconstruction of unprecedented scale, and it suggests that the reconstruction of the tree of life with its 1.7 million species is computationally feasible. The accuracy of supertree methods has often been criticized. In Chapter 4, we presented novel methods that can significantly improve on estimated supertrees and compute larger exact supertrees. These methods present new computational problems that we solve with dynamic programming. We demonstrated the outstanding performance by improving on the best RF supertree by up to 5% and computing exact supertrees of up to 17 taxa. Our methods are not limited to the RF distance, but can also be applied to various other measures, including gene duplications. Despite our impressive results, much more research needs to be accomplished to ensure high quality, large-scale supertrees. Although our implementation allows for construction of large-scale supertrees with up to 1000 taxa, it requires additional algorithmic improvements to scale towards the tree of life. The scale and complexity of modern gene/species tree reconciliations makes detailed analyses difficult. My work on interactive tree analysis and exploration provides a novel method combined with an intuitive design that allows for the first time an interactive analysis of gene duplication, gene loss, and deep coalescence across multiple large-scale species and gene trees. The interactive tree analysis is extendable and it should be interesting to see what additional features can enrich the interactive analysis.

BIBLIOGRAPHY

[1] B. L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combintorics*, 5:1–13, 2001. 8, 10, 30

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Mathematical Biology*, 215:403–410, 1990. 21

[3] D. A. Bader, U. Roshan, and A. Stamatakis. Computational grand challenges in assembling the tree of life: Problems & solutions. *Advances in Computing in Computational Biology and Bioinformatics*, 68:127–176, 2006. 7

[4] M. S. Bansal, J. G. Burleigh, and O. Eulenstein. Efficient genome-scale phylogenetic analysis under the duplication-loss and deep coalescence cost models. *BMC bioinformatics*, 11(S-1):42, 2010. 8, 24, 63

[5] M. S. Bansal, J. G. Burleigh, O. Eulenstein, and D. Fernández-Baca. Robinson-Foulds supertrees. *Algorithms for Molecular Biology*, 5:18, 2010. 3, 12, 47, 48, 49, 50, 59

[6] M. S. Bansal, J. G. Burleigh, O. Eulenstein, and A. Wehe. Heuristics for the gene-duplication problem: A $\theta(n)$ speed-up for the local search. In T. Speed and H. Huang, editors, *Conference on Research in Computational Molecular Biology*, volume 4453 of *Lecture Notes in Computer Science*, pages 238–252. Springer, 2007. 2, 8, 12, 20, 27, 28, 30, 31, 32, 36, 37

[7] M. S. Bansal and O. Eulenstein. An $\theta(n^2/log\ n)$ speed-up of TBR heuristics for the gene-duplication problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 514–524, 2008. 2, 9, 18, 27

[8] M. S. Bansal, O. Eulenstein, and A. Wehe. The gene-duplication problem: Near-linear time algorithms for NNI based local searches. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):221–231, 2009. 18, 63

[9] M. S. Bansal and R. Shamir. A note on the fixed parameter tractability of the gene-duplication problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(3):848–850, 2011. 18

[10] J. P. Barthelemy and F. R. McMorris. The median procedure for n-trees. *Journal of Classification*, 3:329–334, 1986. 47, 48, 50

[11] R. G. Beiko, W. F. Doolittle, and R. L. Charlebois. The impact of reticulate evolution on genome phylogeny. *Systematic Biology*, 57:844–856, 2008. 7

[12] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In G. H. Gonnet, D. Panario, and A. Viola, editors, *Latin American Theoretical Informatics*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. 11, 30, 70

[13] O. R. P. Bininda-Emonds. The evolution of supertrees. *Trends in Ecology & Evolution*, 19:315–322, 2004. 47

[14] O. R. P. Bininda-Emonds, J. L. Gittleman, and M. A. Steel. The (super) tree of life: Procedures, problems, and prospects. *Annual Review of Ecology and Systematics*, 33:265–289, 2002. 47

[15] O. R. P. Bininda-Emonds and M. J. Sanderson. Assessment of the accuracy of matrix representation with parsimony analysis supertree construction. *Systematic Biology*, 50:565–579, 2001. 3

[16] P. Bonizzoni, G. D. Vedova, and R. Dondi. Reconciling gene trees to a species tree. In R. Petreschi, G. Persiano, and R. Silvestri, editors, *Italian Conference on Algorithms and Complexity*, volume 2653 of *Lecture Notes in Computer Science*, pages 120–131. Springer, 2003. 28

[17] M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combintorics*, 8:409–423, 2004. 8, 10, 30

[18] J. G. Burleigh, M. S. Bansal, O. Eulenstein, S. Hartmann, A. Wehe, and T. J. Vision. Genome-scale phylogenetics: Inferring the plant tree of life from 18,896 discordant gene trees. *Systematic Biology*, 60(2):117–125, 2011. 8, 20, 75

[19] J. G. Burleigh, W. B. Barbazuk, J. M. Davis, A. M. Morse, and P. S. Soltis. Exploring diversification and genome size evolution in extant gymnosperms through phylogenetic synthesis. *Journal of Botany*, 292857, 2012. 59

[20] M. Cardillo, O. R. P. Bininda-Emonds, E. Boakes, and A. Purvis. A species-level phylogenetic supertree of marsupials. *Journal of Zoology*, 264:11–31, 2004. 59

[21] W. Chang, J. G. Burleigh, D. F. Fernández-Baca, and O. Eulenstein. An ILP solution for the gene duplication problem. *BMC bioinformatics*, 12(Suppl 1):S14, 2011. 48

[22] R. Chaudhary, M. S. Bansal, A. Wehe, D. Fernández-Baca, and O. Eulenstein. iGTP: A software package for large-scale gene tree parsimony analysis. *BMC bioinformatics*, 11:574, 2010. 63, 65

[23] R. Chaudhary, J. G. Burleigh, and D. Fernández-Baca. Fast local search for unrooted Robinson-Foulds supertrees. In J. Chen, J. Wang, and A. Zelikovsky, editors, *International Symposium of Bioinformatics Research and Applications*, volume 6674 of *Lecture Notes in Computer Science*, pages 184–196. Springer, 2011. 3, 12, 47, 48, 50

[24] K. Chen, D. Durand, and M. Farach-Colton. Notung: A program for dating gene duplications and optimizing gene family trees. *Journal of Computational Biology*, 7:429–447, 2000. 28, 62, 63

[25] T. J. Davies, S. A. Fritz, R. Grenyer, C. D. L. Orme, J. Bielby, O. R. P. Bininda-Emonds, M. Cardillo, K. E. Jones, J. L. Gittleman, G. M. Mace, and A. Purvis. Phylogenetic trees and the future of mammalian biodiversity. *Proceedings of the National Academy of Sciences of the United States of America*, 105:11556–11563, 2008. 6, 46

[26] J. P. Doyon, Ranwez V., V. Daubin, and V. Berry. Models, algorithms and programs for phylogeny reconciliation. *Briefings in Bioinformatics*, 12 (5):392–400, 2011. 62

[27] Z. Du, F. Lin, and U. W. Roshan. Reconstruction of large phylogenetic trees: A parallel approach. *Computational Biology and Chemistry*, 29:273–280, 2005. 27

[28] R. C. Edgar. Muscle: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32:1792–1797, 2004. 21

[29] E. J. Edwards, C. J. Still, and M. J. Donoghue. The relevance of phylogeny to studies of global change. *Trends in Ecology & Evolution*, 22:243–249, 2007. 46

[30] B. C. Emerson and R. G. Gillespie. Phylogenetic analysis of community assembly and structure over space and time. *Trends in Ecology & Evolution*, 23:619–630, 2008. 6

[31] O. Eulenstein. *Predictions of gene-duplications and their phylogenetic development*. PhD thesis, University of Bonn, Germany, 1998. GMD Research Series No. 20 / 1998, ISSN: 1435-2699. 28

[32] O. Eulenstein, D. Chen, J. G. Burleigh, D. Fernández-Baca, and M. J. Sanderson. Performance of flip supertree construction with a heuristic algorithm. *Systematic Biology*, 53:299–308, 2004. 3

[33] M. R. Fellows, M. T. Hallett, and U. Stege. Analogs & duals of the mast problem for sequences & trees. *Journal of Algorithms*, 49(1):192–216, 2003. 3, 18, 30

[34] J. Felsenstein. Inferring phylogenies. *Sinauer Associates, Sunderland, Massachusetts*, 2004. 1

[35] F. Forest, R. Grenyer, M. Rouget, T. J. Davies, R. M. Cowling, D. P. Faith, A. Balmford, J. C. Manning, S. Proches, M. van der Bank, G. Reeves, T. A. J. Hedderson, and V. Savolainen. Preserving the evolutionary potential of floras in biodiversity hotspots. *Nature*, 445:757–760, 2007. 46

[36] P. A. Goloboff, S. A. Catalano, J. Marcos Mirande, C. A. Szumik, J. Salvador Arias, M. Källersjö, and J. S. Farris. Phylogenetic analysis of 73 060 taxa corroborates major eukaryotic groups. *Cladistics*, 25(3):211–230, 2009. 7

[37] M. Goodman, J. Czelusniak, G. W. Moore, A. E. Romero-Herrera, and G. Matsuda. Fitting the gene lineage into its species lineage. a parsimony strategy illustrated by cladograms constructed from globin sequences. *Systematic Zoology*, 28:132–163, 1979. 2, 7, 8, 10, 17, 27, 28

[38] P. Górecki and J. Tiuryn. On the structure of reconciliations. In J. Lagergren, editor, *Comparative Genomics*, volume 3388 of *Lecture Notes in Computer Science*, pages 42–54. Springer, 2004. 28

[39] R. Guigó, I. Muchnik, and T. F. Smith. Reconstruction of ancient molecular phylogeny. *Molecular Phylogenetics and Evolution*, 6(2):189–213, 1996. 7, 27, 28

[40] M. T. Hallett and J. Lagergren. New algorithms for the duplication-loss model. In R. Shamir, S. Miyano, S. Istrail, P. Pevzner, and M. S. Waterman, editors, *Conference on Research in Computational Molecular Biology*, pages 138–146, 2000. 30, 52

[41] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. 30

[42] S. Hartmann, D. Lu, J. Phillips, and T. J. Vision. Phytome: A platform for plant comparative genomics. *Nucleic Acids Research*, 34:D724–D730, 2006. 60

[43] P. H. Harvey and M. D. Pagel. The comparative method in evolutionary biology. *Oxford University Press, New York*, 1991. 6

[44] D. T. Jones, W. R. Taylor, and J. M. Thornton. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences*, 8:25–282, 1992. 60

[45] B. Ma, M. Li, and L. Zhang. On reconstructing species trees from gene trees in term of duplications and losses. In S. Istrail, P. Pevzner, and M. S. Waterman, editors, *Conference on Research in Computational Molecular Biology*, pages 182–191, 1998. 27, 30

[46] B. Ma, M. Li, and L. Zhang. From gene trees to species trees. *SIAM Journal on Computing*, 30:729–752, 2000. 2, 8, 18, 47

[47] W. P. Maddison. Gene trees in species trees. *Systematic Biology*, 46:523–536, 1997. 7

[48] W. P. Maddison and D.R. Maddison. Mesquite: a modular system for evolutionary analysis. version 2.75 http://mesquiteproject.org, 2011. 63

[49] B. Mirkin, I. Muchnik, and T. F. Smith. A biology consistent model for comparing molecular phylogenies. *Journal of Computational Biology*, 2(4):493–507, 1995. 28

[50] K. C. Nixon. The parsimony ratchet: A new method for rapid parsimony analysis. *Cladistics*, 15:407–414, 1999. 24

[51] M. Ott, J. Zola, S. Aluru, and A. Stamatakis. Large-scale maximum likelihood-based phylogenetic analysis on the IBM BlueGene/L. *Proceeding IEEE/ACM Supercomputing Conference*, 2007. 7

[52] Roderic D. M. P. Genetree. `http://taxonomy.zoology.gla.ac.uk/rod/genetree/genetree.html`. 30

[53] R. D. M. Page. Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. *Systematic Biology*, 43(1):58–77, 1994. 28

[54] R. D. M. Page. Genetree: comparing gene and species phylogenies using reconciled trees. *Bioinformatics*, 14:819–820, 1998. 18, 23, 27, 28, 62, 64, 65

[55] R. D. M. Page. Extracting species trees from complex gene trees: Reconciled trees and vertebrate phylogeny. *Molecular Phylogenetics and Evolution*, 14:89–106, 2000. 2, 27

[56] R. D. M. Page and M. A. Charleston. From gene to organismal phylogeny: Reconciled trees and the gene tree/species tree problem. *Molecular Phylogenetics and Evolution*, 7:231–240, 1997. 2

[57] R. D. M. Page and J. Cotton. Vertebrate phylogenomics: Reconciled trees and gene duplications. In *Proceedings of Pacific Symposium on Biocomputing 2002 (PSB2002)*, pages 536–547, 2002. 27

[58] D. Penny, W. T. White, M. D. Hendy, and M. J. Phillips. A bias in ML estimates of branch lengths in the presence of multiple signals. *Molecular Biology and Evolution*, 25:239–242, 2008. 7

[59] R. E. Ricklefs. Estimating diversification rates from phylogenetic information. *Trends in Ecology & Evolution*, 22:601–610, 2007. 6

[60] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981. 1, 3, 50

[61] M. J. Sanderson. r8s: Inferring absolute rates of molecular evolution and divergence times in the absence of a molecular clock. *Bioinformatics*, 19:301–302, 2003. 22

[62] M. J. Sanderson, A. Purvis, and C. Henze. Phylogenetic supertrees: Assembling the trees of life. *Trends in Ecology & Evolution*, 13:105–109, 1998. 47

[63] B. Sennblad, E. Schreil, A. B. Sonnhammer, J. Lagergren, and L. Arvestad. Primetv: A viewer for reconciled trees. *BMC bioinformatics*, 8:148, 2007. 62, 63, 64

[64] J. B. Slowinski, A. Knight, and A. P. Rooney. Inferring species trees from gene trees: A phylogenetic analysis of the elapidae (serpentes) based on the amino acid sequences of venom proteins. *Molecular Phylogenetics and Evolution*, 8:349–362, 1997. 27

[65] A. Stamatakis. RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22:2688–2690, 2006. 21, 60

[66] A. Stamatakis, P. Hoover, and J. Rougemont. A fast bootstrapping algorithm for the RAxML web-servers. *Systematic Biology*, 57(5):758–771, 2008. 2, 48

[67] A. Stamatakis, T. Ludwig, and H. Meier. RAxML-II: A program for sequential, parallel & distributed inference of large phylogenetic trees. *Concurrency and Computation: Practice and Experience*, 17:1705–1723, 2003. 27

[68] U. Stege. Gene trees and species trees: The gene-duplication problem in fixed-parameter tractable. In F. K. H. A. Dehne, A. Gupta, J. Sack, and R. Tamassia, editors, *International Workshop on Algorithms and Data Structures*, volume 1663 of *Lecture Notes in Computer Science*, pages 288–293. Springer, 1999. 30

[69] D. L. Swofford. PAUP*: Phylogenetic analysis using parsimony (*and other methods), version 4.0b10, 2002. 2, 48

[70] J. S. Taylor and J. Raes. Duplication and divergence: The evolution of new genes and old ideas. *Annual Review of Genetics*, 38:615–643, 2004. 2

[71] C. Than and L. Nakhleh. Species tree inference by minimizing deep coalescences. *PLoS Comput Biol*, 5(9):e1000501, 09 2009. 48, 52

[72] W. Thuiller, S. Lavergne, C. Roquet, I. Boulangeat, B. Lafourcade, and M.B. Araujo. Consequences of climate change on the tree of life in Europe. *Nature*, 470:531–534, 2011. 46

[73] A. Wehe, M. S. Bansal, J. G. Burleigh, and O. Eulenstein. DupTree: A program for large-scale phylogenetic analyses using gene tree parsimony. *Bioinformatics*, 24(13):1540–1541, 2008. 12, 18, 20, 23, 48

[74] A. Wehe and J. G. Burleigh. Scaling the gene duplication problem towards the tree of life. *Bioinformatics and Computational Biology*, 2010. 4, 63

[75] A. Wehe, J. G. Burleigh, and O. Eulenstein. Algorithms for knowledge-enhanced supertrees. in press, 2012. 4

[76] A. Wehe, W. Chang, O. Eulenstein, and S. Aluru. A scalable parallelization of the gene duplication problem. *Journal of Parallel and Distributed Computing*, 70(3):237–244, 2010. 4, 24

[77] M. Wilkinson, J. A. Cotton, C. Creevey, O. Eulenstein, S. R. Harris, F. J. Lapointe, C. Levasseur, J. O. McInerney, D. Pisani, and J. L. Thorley. The shape of supertrees to come: Tree shape related properties of fourteen supertree methods. *Systematic Biology*, 54:419–432, 2005. 3

[78] C. G. Willis, B. Ruhfel, R. B. Primack, A. J. Miller-Rushing, and C. C. Davis. Phylogenetic patterns of species loss in Thoreauś woods are driven by climate change. *Proceedings of the National Academy of Sciences of the United States of America*, 105:17029–17033, 2009. 46

[79] Y. Yu and L. Warnow, T. Nakhleh. Algorithms for MDC-based multi-locus phylogeny inference: Beyond rooted binary gene trees on single alleles. *Journal of Computational Biology*, 18(11):1543–1559, 2011. 48, 52

[80] L. Zhang. On a Mirkin-Muchnik-Smith conjecture for comparing molecular phylogenies. *Journal of Computational Biology*, 4(2):177–187, 1997. 28, 30

[81] L. Zhang. Inferring a species tree from gene trees under the deep coalescence cost. *Conference on Research in Computational Molecular Biology*, pages 192–193, 2000. 47